Universidad Autónoma de Madrid
Departamento de Ingeniería Informática

# Técnicas de Verificación Térmica para Arquitecturas Dinámicamente Reconfigurables

## TESIS DOCTORAL

Autor: Sergio López Buedo

Director: Eduardo Boemo Scalvinoni

Escuela Politécnica Superior

Mayo de 2003

*I am not young enough to know everything*

# Resumen

La verificación térmica es una técnica cada vez más útil en el mundo de las FPGAs (*Field-Programmable Gate Arrays*). Varias son las razones para esta afirmación: la creciente capacidad de los dispositivos lógicos programables, el aumento de la complejidad y del consumo de los circuitos que se implementan en ellos, y la aparición de nuevos estilos de diseño, como la reconfiguración en tiempo de ejecución. En esta tesis se han estudiado diferentes posibilidades para integrar sensores de temperatura en FPGAs: osciladores en anillo, diodos de enclavamiento y osciladores dedicados, embebidos por el fabricante de los dispositivos. Se ha realizado una experimentación exhaustiva, y se ha desarrollado una metodología para la operación y el calibrado que elimina el problema del autocalentamiento. De todos los sensores, los osciladores en anillo han resultado ser la mejor opción: son pequeños y sencillos, se pueden implementar en cualquier posición del chip y en cualquier dispositivo programable, y tienen muy buena linealidad y una buena sensibilidad, del orden de 0,2% por °C. Empleando esta técnica, se ha creado un sensor de temperatura para arquitecturas reconfigurables en tiempo de ejecución, que puede ser dinámicamente insertado y eliminado de la FPGA, sin alterar el funcionamiento del resto del circuito. Adicionalmente, se ha desarrollado un método que elimina la mayor desventaja de los osciladores en anillo: su sensibilidad a las variaciones en la tensión de alimentación. Por último, se ha demostrado que con esta técnica es posible realizar mapas térmicos de FPGAs en condiciones reales de funcionamiento, una aplicación inédita, imposible de realizar con las herramientas disponibles hasta el momento. Entre sus principales utilidades figuran la detección de zonas calientes dentro del circuito, la cuantificación de la eficacia de una determinada opción de refrigeración, la detección de errores por reconfiguración o la localización de fallos de diseño tales como colisiones de buses. Se abre además un nuevo campo de aplicación de las FPGAs: estos dispositivos pueden resultar de gran utilidad para todos aquellos diseñadores que se ocupan de medir los parámetros térmicos de los encapsulados de los circuitos integrados..

II

# Abstract

Thermal verification is an increasingly useful technique in the field of FPGAs (Field-Programmable Gate-Arrays). There are some reasons for this statement: the increasing capacity of programmable logic devices, the growing capacity and power consumption of the circuits implemented on them, and the development of new design styles, like run-time reconfiguration. In this thesis several possibilities for FPGA-integrated temperature sensors have been studied: ring-oscillators, clamping diodes and dedicated oscillators, embedded by the manufacturer. Among all sensors, ring-oscillators were found to be the best option: they are small and simple, can be placed on any position of the chip in every programmable logic device, and have a good linearity and sensitivity, around 0,2% per °C. By using this technique, a temperature sensor for run-time reconfigurable architectures has been developed, which can be dynamically inserted and eliminated from the FPGA without altering the rest of the circuit. Moreover, a method to eliminate the influence of power supply variations has been developed: this is the major disadvantage of ring-oscillators. Finally, this technique has proved to be useful to make thermal maps of an FPGA in real working conditions, a completely new application, impossible to be made with the existing tools. Among its main benefits are applications like detecting hot-spots in the circuit, the quantification of a given cooling option, identifying reconfiguration errors or finding design flaws like bus contentions. Additionally, this thesis opens a new application field for FPGAs: these devices might be very useful for those designers responsible for measuring the thermal parameters of integrated circuits.

# Agradecimientos

Durante todos estos años, Eduardo ha sido mucho más que un director de tesis; su amistad, conocimientos, confianza, complicidad y entusiasmo permanente han sido insustituibles, no sólo para esta investigación, sino para toda mi carrera académica.

Esta tesis no hubiera sido posible sin la amistad, el apoyo y el buen ambiente creado por mis compañeros de laboratorio; quiero agradecer a Javier (y también a Javier), y a Paco su continua confianza y colaboración, a Guillermo su amistad y ayuda en tantos proyectos, a Gustavo sus valiosos conocimientos sobre FPGAs, casi tanto como sus asados, a Iván y Elías por sus siempre oportunas sugerencias y comentarios, a Rubén por ayudarme a conseguir mucha de la música que amenizó la escritura de esta tesis, a Juan por su ayuda en los proyectos y por hacerme un poco más linuxero, a Miguel por su amistad y sus divertidas anécdotas empresariales, y a Antonio y Susana por su respaldo en la docencia, cuando esta investigación no me dejaba dedicarla demasiado tiempo. Y por supuesto a Antonio, que antes de volverse al sur fue quien me dio los ánimos definitivos para terminar esta tesis.

Los comienzos de esta investigación fueron en el Laboratorio de Microelectrónica de la Facultad de Ciencias, y para muchos de los experimentos he utilizado sus equipos; quiero agradecer a todos sus integrantes esta ayuda, sin la cual no hubiera sido posible realizarlos, aunque lo más importante para mí fue su amistad y calidez. A Basilio y José Luis por tantas amenas comidas y tan buenos y constructivos comentarios, a Juan por su apoyo y confianza, a Pablo por su colaboración en esta investigación, y por su simpatía (y su acento), a Pedro y Edu por su ayuda incondicional en los experimentos, siempre dispuestos a prestarme parte de su tiempo, y a Marichu, Manolo y Lucía por su amistad y compañerismo.

No me puedo olvidar de Juana, sin ella no hubiese conseguido vencer a la burocracia, ni de Ángel ni Eugenio, sin su ayuda en los laboratorios de alumnos nunca habría tenido tiempo para acabar esta tesis.

Para terminar, quisiera dar las gracias a las empresas que colaboraron con nosotros, pues con estos proyectos se financiaron los inicios de esta tesis. Quiero agradecer a Roberto, Fernando y Carlos de Fedetec, y a Carlos y Paco de Pedro Sanz Clima su colaboración, comprensión y buen talante.

# Índice

# Índice de figuras

# Índice de tablas

# Capítulo 1.

# Introducción y objetivos de la tesis

## 1. Introducción al concepto de verificación térmica

Según el Diccionario de la Real Academia, verificar es "probar que una cosa que se dudaba es verdadera", y este es el objetivo final de las técnicas que se presentarán en esta tesis: probar (y asegurar) a través de medidas de temperatura que un sistema electrónico está funcionando correctamente. En particular, la tesis se centra en los sistemas basados en lógica programable, y más concretamente, en los dinámicamente reconfigurables.

La necesidad de la verificación térmica viene por dos caminos. En primer lugar, los circuitos electrónicos sólo pueden funcionar en un margen de temperaturas relativamente reducido (por ejemplo, Xilinx aconseja no superar los 125 °C). Estos límites se pueden superar fácilmente, no sólo porque las condiciones ambientales sean desfavorables, sino porque en muchos casos el chip no es capaz de disipar el calor producido por sus propios circuitos internos.

El segundo de los motivos para la verificación térmica es que muchos de los errores en los circuitos electrónicos causan incrementos en la potencia consumida, que se traducen en un aumento de la temperatura. Estos incrementos anormales de la temperatura no sólo indican un funcionamiento incorrecto, sino que además pueden provocar daños permanentes.

De esta manera, la verificación térmica aparece como una técnica con doble utilidad: primero, para probar que el circuito puede funcionar correctamente porque su

temperatura está dentro de los límites especificados, y segundo, para asegurar que en efecto lo está haciendo, porque sus patrones de consumo (y por tanto de temperatura) están dentro de lo esperado.

Entrando más en detalle, es evidente que el consumo de los circuitos integrados se incrementa cada vez más, y todo indica que este camino va a continuar en los próximos años [SIA00]. Atrás quedó la época en que los complicados sistemas de refrigeración sólo se veían en los superordenadores o en otros sistemas muy especializados. Hoy en día algo tan común como un ordenador portátil se ha convertido en un complejísimo sistema desde el punto de vista térmico, que necesita de una gran variedad de componentes para refrigerarse: distintos tipos de disipadores, varios ventiladores, tuberías de calor... Y además, la ingeniería de estos sistemas ya no es un proceso sencillo, que pueda hacerse sólo con sumas y multiplicaciones: necesita de complicadas herramientas de cálculo diferencial basado en elementos finitos.

Por otro lado, durante años la temperatura ha sido un parámetro primario en las técnicas de depuración del hardware electrónico. En el pasado, la baja temperatura identificaba a tubos de vacío funcionando incorrectamente. Hoy, circuitos integrados que queman (T > 60 °C) apuntan a problemas potenciales. Incluso en una era de instrumentación compleja, los diseñadores de hardware aún conservan el hábito de descubrir circuitos defectuosos tocando sus encapsulados. De este modo, la tendencia creciente de incluir sensores de temperatura en los sistemas electrónicos actuales no ha sido más que un paso natural. Sí, el concepto de "comprobación termal" o *thermal testing* [Sze95] ha sido incorporado en varios productos electrónicos.

Finalmente, existe una consideración adicional, que es la fiabilidad: una elevada temperatura de operación influye muy negativamente en la vida de los circuitos integrados. Este es un tema muy complejo, donde no valen reglas del tipo "10 °C menos doblan la vida del circuito" [Pec97]. Lo que es cierto es que un diseño que requiera operar durante muchos años debe trabajar a una temperatura no sólo moderada en media, sino también sin variaciones bruscas ni gradientes intensos.

# 2. Verificación térmica en el contexto de arquitecturas dinámicamente reconfigurables

Los sistemas basados en lógica programable presentan una serie de características que los convierten en muy buenos candidatos para la verificación térmica. En primer lugar está la inherente versatilidad de las FPGAs. Un microprocesador puede variar su consumo dependiendo del programa que esté ejecutando, por lo que habrá dimensionar su disipador de tal manera que pueda manejar la máxima potencia. Pero esto mismo no es válido para los circuitos programables: no tiene sentido pensar en un disipador universal que valga para todos los diseños que puedan implementarse en una FPGA. Si se hiciera así, sería una gran pérdida de recursos, porque a diferencia de los microprocesadores, las diferencias de consumo que se pueden tener son muy grandes, incluso de hasta dos órdenes de magnitud o más. Cada diseño deberá tener entonces su propio solución de refrigeración, dependiendo de la potencia que consuma.

Lo anterior obliga al diseñador que trabaje con FPGAs a saber dimensionar el disipador que debe utilizar su circuito. Esta tarea no es fácil: tanto las herramientas de estimación de consumo como los modelos térmicos distan mucho de ser perfectos, como se verá en el siguiente capítulo. En los diseños estáticos este problema se puede paliar mediante el método de la prueba y error, haciendo experimentos en el laboratorio hasta dar con el disipador adecuado. Pero estas pruebas no se pueden hacer en los sistemas dinámicamente reconfigurables, simplemente porque el diseño final no se conoce. Y estos sistemas no sólo pertenecen al campo de la investigación, sino que pueden ser algo tan común como una FPGA en una tarjeta para PC cuyo *bitstream* se actualiza al cambiar el *driver*. Como prueba de esto, diversas protecciones térmicas han sido adoptadas en varias FCCMs (*FPGA-Based Custom Computing Machines*) comerciales [Vcc99, Gig97]. Adicionalmente, las FPGAs de mayores prestaciones tanto de Xilinx como de Altera incluyen sensores integrados de temperatura, como se verá en el capítulo 3.

Otra característica que complica el diseño térmico de los sistemas basados en lógica programable es el tamaño de los dispositivos. Hoy en día, las FPGAs tienen millones de puertas disponibles, por lo que se pueden construir en ellas verdaderos "sistemas en un chip", que incluyen circuitos muy dispares tanto en funcionalidad como en consumo.

Como consecuencia de esto, una elevada temperatura de operación puede ser causada no por un consumo global alto, sino por un bloque particular. No tendrá sentido entonces rediseñar todo el chip, cuando se puede actuar sólo sobre el causante de este incremento y mantener la funcionalidad del resto de componentes. Pero esto obliga al desarrollo de estrategias de verificación térmica mucho más complejas, más allá de la simple inclusión de un sensor integrado en el chip.

Por otro lado, las arquitecturas dinámicamente reconfigurables son propensas a errores que se manifiestan con un incremento en la disipación de potencia, como fallos de configuración o colisiones de buses. En estos sistemas, los distintos cores que se integran en la FPGA se cargan y descargan dinámicamente, bajo demanda, de tal manera que puede haber una gran variedad de posibles configuraciones. Bajo ese supuesto, va a ser muy difícil hacer un análisis *a priori* que elimine la posibilidad de fallos en todas las combinaciones. Una estrategia de verificación térmica puede ayudar a detectar estos errores, por ejemplo si se mide un incremento de temperatura inesperado al cargar un determinado módulo.

Lo anterior es especialmente cierto en los circuitos evolutivos [Sip99], que se basan en *bitstreams* que van mutando hasta que se consigue un diseño con la funcionalidad deseada, haciendo un paralelismo con la evolución de las especies en biología. Si este proceso no se realiza con el cuidado suficiente, en estas mutaciones pueden aparecer circuitos con errores que causan un gran consumo de potencia, y que por tanto provocar la destrucción de la FPGA.

## 3. Objetivos de la tesis

Como conclusión de todo lo anterior, el objetivo de esta tesis es desarrollar una serie de técnicas que permitan implementar estrategias de verificación térmica en circuitos programables. La meta no sólo será crear sensores de temperatura integrados, fundamentales para la verificación térmica, sino también realizar experimentos que traten de evaluar si la información proporcionada por estos sensores será útil para detectar posibles fallos, o para conocer cual es el consumo de los distintos bloques implementados en la FPGA.

Esta tesis tiene un marcado carácter experimental, y se ha desarrollado sobre dispositivos del fabricante Xilinx. Sin embargo, esto no significa que sus resultados no

sean extrapolables a otros fabricantes o a dispositivos futuros. La uniformidad de los resultados obtenidos, aún empleando FPGAs con más de 10 años de diferencia de edad, indican que los sensores desarrollados en esta tesis pueden ser implementados en prácticamente cualquier dispositivo programable.

En resumen, los principales objetivos de esta tesis pueden dividirse en tres grandes grupos:

**1. Estudio e implementación de las diferentes posibilidades de sensores de temperatura integrados para FPGAs:** este debe ser el inicio de cualquier estrategia de verificación térmica. Poco se puede hacer si no se disponen de sensores integrados en el chip. Como se verá en el siguiente capítulo, la estimación de la temperatura del silicio a partir de medidas en el encapsulado o en el ambiente es un camino lleno de incertidumbres, cuyos resultados serán cuanto menos dudosos.

**2. Investigación del sensado de temperatura en el contexto de arquitecturas dinámicamente reconfigurables:** ya se ha visto que estas arquitecturas son muy buenas candidatas para la verificación térmica, pero es que el inverso también es cierto: la medida de la temperatura es una aplicación muy adecuada para ser manejada con reconfiguración dinámica.

La configuración más sencilla utilizará un único sensor por chip; de esta manera se podrá saber si el circuito está funcionando dentro de los límites especificados. Puesto que las constantes térmicas dominantes son mucho mayores que el tiempo de reconfiguración, el sensor se podría añadir dinámicamente cuando se quiera tomar una medida, y luego se eliminará para dejar espacio a otra aplicación. Pero si la lectura de la temperatura es anormal, lo que interesará es añadir un conjunto de sensores, para tratar de saber que está pasando a partir de los gradientes de temperatura que se detecten. Pero no tendrá sentido mapear este conjunto de sensores estáticamente, porque su información sólo se utilizará en casos excepcionales.

El objetivo será entonces ver si es posible crear un sensor de temperatura que pueda ser dinámicamente insertado y eliminado de la FPGA, en tiempo de ejecución, sin alterar el funcionamiento del resto de circuitos implementados en el dispositivo.

**3. Evaluación de la posibilidad y utilidad de crear mapas térmicos de dispositivos programables en funcionamiento:** los puntos anteriores tratan de la

posibilidad de medir la temperatura del silicio en los dispositivos programables; la aplicación más común será tomar la medida en un único punto. Aunque esta información es muy importante, porque con ella se puede asegurar que el circuito está operando dentro de los limites especificados, el objetivo final debe ser más ambicioso. Y esta meta debe ser obtener información acerca del funcionamiento del chip a través de los gradientes de temperatura que ocurran en él

El primer paso será comprobar si es posible realizar mapas térmicos de una FPGA en funcionamiento, sin alterar su normal operación. Y una vez que se hayan desarrollado las técnicas que faciliten esta posibilidad, averiguar si en efecto existen gradientes de temperatura en el chip, y si estos gradientes verdaderamente informan de que bloques están consumiendo más potencia. Para ello se deberán emplear tanto circuitos de prueba, que generen puntos calientes (*hot-spots*) en el dado, como diseños reales, que demuestren la utilidad práctica de esta técnica.

## 4. Organización de la tesis

Esta tesis está organizada en 7 capítulos, a los que se agregan dos apéndices. En el primero se hace una introducción práctica al flujo de diseño escogido para manejar la reconfiguración en tiempo de ejecución: JBits. En el segundo apéndice se detallan los desarrollos y las tarjetas que se utilizaron para realizar los experimentos, y se presenta el código de un sensor de temperatura dinámicamente insertable en la FPGA. Toda esta información puede resultar de utilidad para otros grupos de investigación que se inicien en esta línea.

En el capítulo 2 se revisan los principales conceptos relacionados con los aspectos térmicos de los circuitos integrados. Esta información se encuentra fragmentada en innumerables manuales y hojas de datos de circuitos comerciales. Dentro del conocimiento del autor, no existe una información unificada sobre este tema.

El capítulo 3 presenta las principales opciones de sensores de temperatura orientados a circuitos integrados. Se cubre tanto el estado del arte (2003) de los circuitos comerciales como las principales ideas desarrolladas en universidades y centros de investigación.

En el capítulo 4 se exponen las ideas centrales desarrolladas en esta tesis. Se muestran las principales características de cada una de las alternativas analizadas, se describe la metodología de calibración y se resumen los aspectos positivos y negativos de cada solución.

El capítulo 5 muestra una serie de aplicaciones ideadas para demostrar la validez de los métodos propuestos. En particular, se desarrolla en detalle una propuesta de combinación entre reconfiguración dinámica y análisis térmico, una de las aportaciones originales de esta Tesis.

En el capítulo 6 se hace un estudio acerca de la viabilidad de obtener mapas térmicos de una FPGA en funcionamiento, sin afectar al resto de circuitos que pudieran estar operando en ella. También se evalúa la utilidad de estos mapas a la hora proporcionar información acerca de que componente está consumiendo más potencia, tanto para puntos calientes diseñados *ad-hoc* como para circuitos reales.

Finalmente, en el capítulo 7 se resumen los principales resultados obtenidos, se describen los principales artículos y aplicaciones producidas por este trabajo de tesis. Como punto final, se describen las líneas futuras de investigación.

# Capítulo 2.

# Aspectos térmicos de los circuitos integrados

En este capítulo se revisan los principales conceptos relacionados con los aspectos térmicos de los circuitos integrados, que como se comentó en la introducción se encuentran repartidos en multitud de enmarañadas notas técnicas y estándares. El objetivo final de este capítulo es demostrar la necesidad de disponer de sensores de temperatura integrados, pues como se verá a continuación ni los modelos actuales de los encapsulados son precisos, ni la potencia se puede estimar adecuadamente; y además los disipadores tampoco son fáciles de modelar, ni completamente predecibles, pues como cualquier otro elemento pueden fallar en un momento dado.

## 1. Modelo térmico del encapsulado de los circuitos integrados

El dato más común (aunque podría decirse que es el único) que ofrecen los fabricantes para caracterizar sus encapsulados es la resistencia térmica. Este es un dato muy valioso, pues predice de una manera muy sencilla la temperatura del silicio:

$$T_J = T_A + \theta_{JA} P_D$$

Donde $T_J$ es la temperatura en la unión (silicio), $T_A$ es la temperatura ambiente, $\theta_{JA}$ es la resistencia térmica desde la unión hasta el ambiente, y $P_D$ es la potencia disipada.

Esta fórmula se obtiene de un modelo similar al de un circuito eléctrico (usando la ley de Ohm), donde la fuente de calor se transforma en una fuente de corriente, las diferencias de temperatura en diferencias de potencial, y las características térmicas de los materiales en resistencias térmicas. En la siguiente figura, obtenida de [TI99], se puede ver cual sería el modelo de un encapsulado típico:

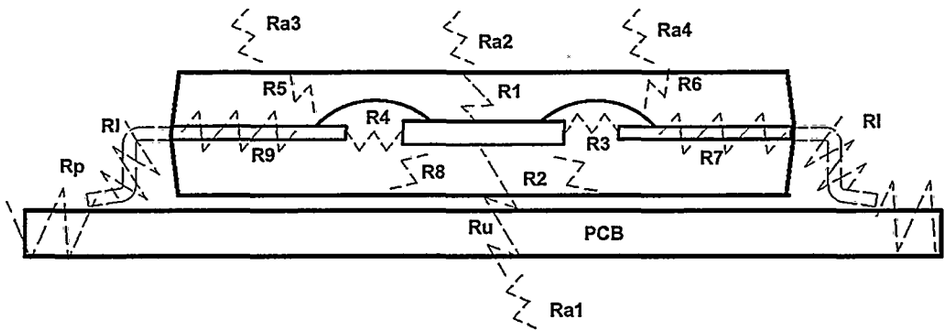



Fig. 1: Resistencias térmicas en un encapsulado típico

Que se correspondería con este circuito equivalente:

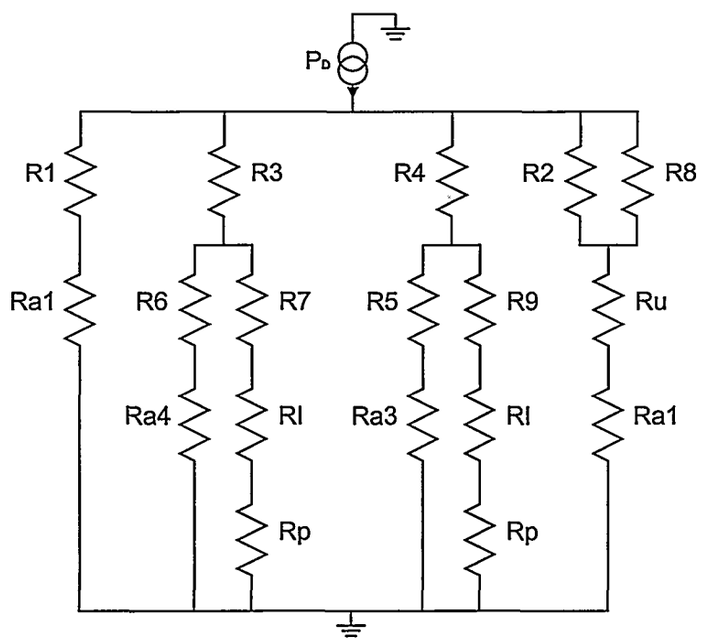



Fig. 2: Circuito equivalente para el modelo térmico de un encapsulado típico

Como ve en la Fig. 1, cada componente del encapsulado tiene una determinada resistencia térmica, de tal manera que la resistencia final $\theta_{JA}$ será el resultado de resolver la red que se obtenga en el circuito equivalente (Fig. 2). Normalmente se suele asociar la temperatura ambiente $T_A$ con la masa del circuito (punto de referencia de tensiones).

Nada más ver este modelo se comprueba que $\theta_{JA}$ incluye la parte de calor disipada a través del circuito impreso. Esto es un problema, porque las características y densidad de pistas en el PCB varían mucho de diseño a diseño. Y estas pistas no son más que segmentos de cobre, un excelente conductor térmico. Como se describe más adelante, las mediciones de la resistencia térmica se hacen utilizando PCBs estandarizados, con una relativamente baja densidad de pistas. De tal manera que es de esperar que el PCB real sea mucho más denso que el usado para caracterizar el encapsulado, por lo que el valor de la resistencia térmica será menor que el ofrecido por el fabricante. En un principio esto puede parecer una ventaja más que un problema, porque asegurará que el circuito estará funcionando siempre dentro de los límites seguros; pero también es cierto que causa que se desperdicien parte de las prestaciones que el circuito podría alcanzar. Y por otro lado, el flujo de aire va a disminuir muy notablemente la resistencia térmica de los encapsulados, como se muestra en la siguiente figura, obtenida de [Xil02]:

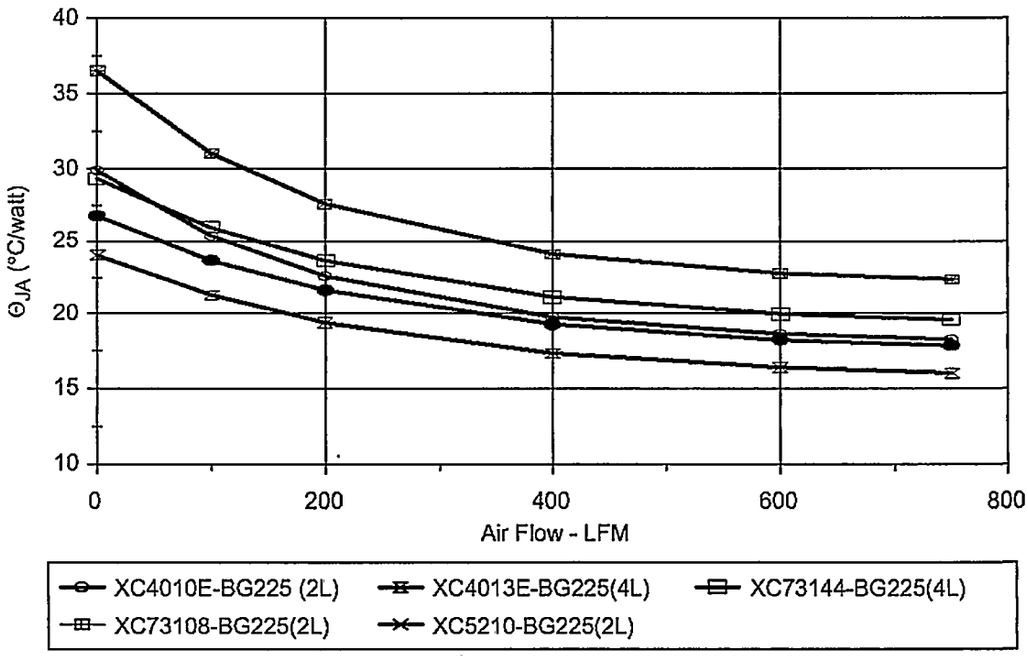



Fig. 3: Influencia del flujo de aire en $\theta_{JA}$ para encapsulados BGA de 225 pines

Como se puede observar en esta figura, la resistencia térmica para FPGAs con un encapsulado muy común (BGA de 225 pines) se puede reducir entre 30 - 40% con sólo aplicar un flujo de aire moderado (200 LFM son aproximadamente 1 $ms^{-1}$)

Como conclusión, el valor de $\theta_{JA}$ debe considerarse sólo como orientativo: existen una gran variedad de factores que capaces de alterar esta cifra. Es por esto que para predecir la temperatura del silicio se prefiere otro parámetro, la resistencia térmica desde la unión hasta el encapsulado, $\theta_{JC}$. Esta resistencia relaciona la temperatura entre estos dos puntos, y así, colocando una sonda de temperatura en la superficie del encapsulado se podría saber la temperatura en la unión. Se emplearía una fórmula similar a la antes mencionada para $\theta_{JA}$:

$$T_J = T_C + \theta_{JC} P_D$$

Además, tradicionalmente se usa esta relación:

$$\theta_{JA} = \theta_{JC} + \theta_{CA}$$

Que expresa que $\theta_{JA}$ está compuesta por una parte fija, dependiente del encapsulado, que es la $\theta_{JC}$ que se acaba de definir, y una parte variable, la resistencia térmica desde el encapsulado hasta el ambiente $\theta_{CA}$, que será lo que varíe con el flujo de aire, o con los disipadores que se añadan... En la siguiente tabla se ponen a modo de ejemplo los valores para las FPGAs utilizadas en esta tesis, más un otros dos en formato BGA [Xil02]:

| Encapsulado | $\theta_{JA}$ Max. 0 LFM | $\theta_{JA}$ Típica 0 LFM | $\theta_{JA}$ Min. 0 LFM | $\theta_{JA}$ Típica 250 LFM | $\theta_{JA}$ Típica 500 LFM | $\theta_{JC}$ Típica |
|---|---|---|---|---|---|---|
| PC84 | 41.7 | 33.3 | 27.9 | 25.8 | 20.8 | 5.6 |
| PQ240 | 28.5 | 19.9 | 14.0 | 14.7 | 13.0 | 3.8 |
| HQ240 | 14.5 | 13.2 | 11.8 | 9.1 | 7.3 | 1.5 |
| FG456 | 23.5 | 19.6 | 16.5 | 15.5 | 14.1 | 2.2 |
| FF896 | 11.8 | 11.8 | 11.8 | 8.2 | 6.7 | 1.1 |

Tabla 1: Resistencias térmicas de encapsulados comunes para FPGAs

Todas las unidades están en $°C \cdot W^{-1}$; PC84 es el código para el encapsulado PLCC de 84 pines, PQ240 y HQ240 corresponden con quad flat packs de 240 pines, la diferencia es que el HQ240 está térmicamente mejorado, FG456 es un BGA de 456 pines, con un *pitch* de 1mm, y FF896 es también un BGA, de 896 pines, con 1mm de *pitch* y montaje *flip-chip*.

Nada más ver esta tabla lo que más sorprende es que no se da un valor concreto para las resistencias térmicas, sino que se dan valores máximos, mínimos y típicos. Esto es porque las cifras dependen del tamaño del dado, como se muestra en la siguiente figura, obtenida también del fabricante [Xil02]:

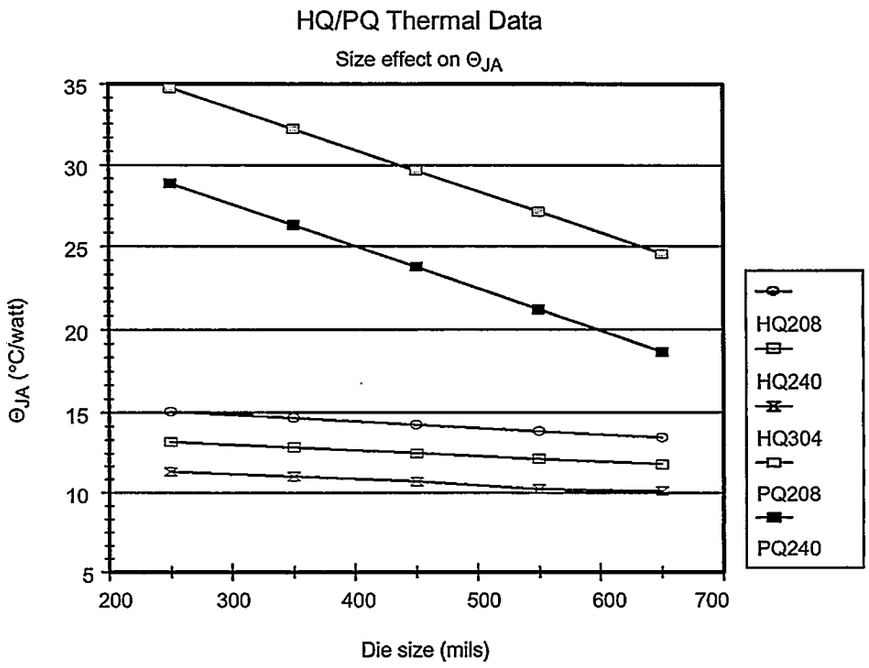



Fig. 4: Efecto del tamaño del dado en la resistencia térmica $\theta_{JA}$

Esta manera de presentar los valores no es uniforme en la industria. Algunos fabricantes [Alt03] ofrecen los datos concretos para cada combinación encapsulado/dispositivo (esto es lo correcto), y sin embargo, otros sólo ofrecen un dato para cada encapsulado [Act03], que de poca ayuda puede resultar. Como conclusión, esta es otra más de las incertidumbres en el valor de la resistencia térmica.

Pero en cualquier caso, todas estos modelos, por muy clásicos que sean, no son correctos. Como se explica en [Las97], tienen muchos puntos débiles. En este artículo se propone como ejemplo este sencillo modelo para el encapsulado:

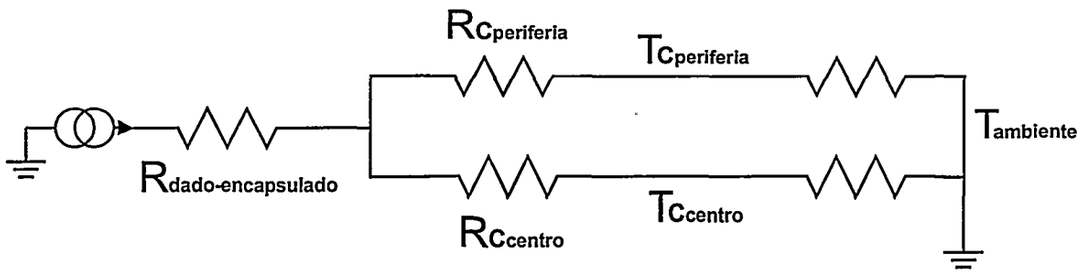



Fig. 5: Ejemplo de modelo de encapsulado incompatible con el cálculo de $\theta_{JC}$

En el que hay dos flujos de calor, y por tanto, dos temperaturas distintas para el centro de encapsulado y para su periferia. Aunque este es un modelo muy sencillo, no hay manera de hacerlo compatible con el clásico, que supone una temperatura uniforme en la superficie del encapsulado:

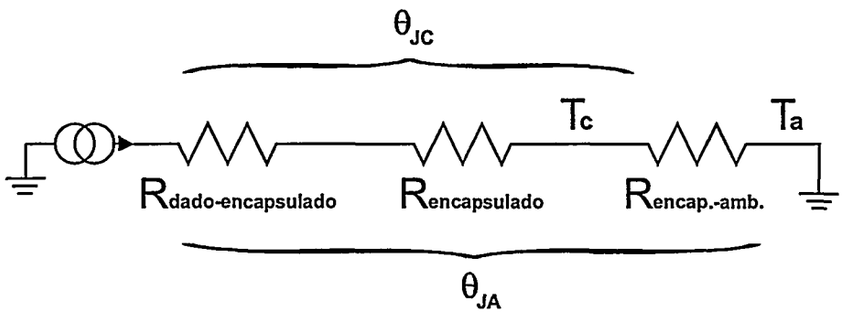



Fig. 6: Modelo clásico para las resistencias térmicas de un encapsulado

$R_{encapsulado}$ sería el paralelo de $R_{Ccentro}$ y $R_{Cperiferia}$. Pero no hay manera de hacer ese paralelo (no tiene sentido físico) si estas dos resistencias tienen un extremo conectado a distintos nodos. Y por supuesto, los modelos reales son todavía más incompatibles, como se puede comprobar volviendo a ver la Fig. 2. El problema surge porque en la realidad los encapsulados no son componentes uniformes, sino sistemas formados por muchos elementos con diferentes características térmicas. Por lo tanto no tiene mucho sentido trabajar con $\theta_{JC}$, pues éste es un valor que, como se verá más adelante, se obtiene en

condiciones irreales, forzando que haya un flujo único de calor e igualando temperaturas en la superficie del encapsulado. Por esta razón, JEDEC en sus estándares [JED95] define un nuevo valor, $\Psi_{JT}$, que se denomina *junction to top thermal parameter*, y que se define como:

$$\psi_{JT} = \frac{T_T - T_J}{P_D}$$

Donde $T_T$ es la temperatura en el centro de la parte superior del encapsulado. Aunque este parámetro no es desde un punto de vista físico una resistencia térmica (que se definen formalmente como la diferencia de temperatura entre dos superficies isotérmicas dividida entre el flujo de calor entre ambas), si que tiene mucho más interés desde el punto de vista práctico, pues este valor es más apropiado para predecir la temperatura del silicio. Sólo en aquellos casos en que haya un único flujo de calor dominante en el encapsulado, que se escape por el centro de su cara superior, $\theta_{JC}$ y $\Psi_{JT}$ convergerán.

Aunque este parámetro sea el más adecuado para predecir la temperatura del silicio, sigue teniendo potenciales inexactitudes. Por ejemplo, su valor depende del flujo de aire. Y esto sin tener en cuenta que la potencia disipada no es predecible actualmente con exactitud, lo que dejaría este método como prácticamente inservible. Y además, existe otro factor que puede hacer que la temperatura suba por encima de la prevista, y es la influencia de puntos calientes cercanos al dispositivo. Estos puntos calientes provocarán que la temperatura sea mayor que la prevista por dos razones: primera, el calor aportado por radiación. Y segunda, el incremento local de la temperatura ambiente que provocarán. No sólo del aire, lo que será especialmente cierto si no hay ventilación forzada, sino también del PCB, pues ya se ha comentado que este es un medio muy relevante en la extracción del calor de un dispositivo.

Como conclusión se puede decir que las resistencias térmicas son datos valiosos para cuantificar lo buenos que son térmicamente los encapsulados, y ya en un segundo lugar, para hacer una estimación grosera de la temperatura de operación. Si un encapsulado tiene una resistencia menor que otro, se puede esperar con poco riesgo de equivocarse que su temperatura de operación sea también menor. Lo que ya no es cierto es que este valor de la resistencia térmica vaya a permitir predecir con una precisión razonable la temperatura en la unión. Y esto también es cierto para el parámetro $\Psi_{JT}$.

## 1.1. Metodología para la obtención de las resistencias térmicas

### *1.1.1. Dados de prueba*

Los valores de las resistencias térmicas no se suelen obtener utilizando el chip real, sino que se emplea habitualmente un dado de prueba. Este circuito contienen uno o varios diodos, para medir la temperatura en la unión, y una resistencia difundida en el silicio, para disipar potencia. La ventajas que tiene usar estos chips es que es mucho más sencillo controlar la potencia que disipan (sólo hay que conectar una fuente regulada a la resistencia), su conexionado es mucho más sencillo que en el caso real, y como un mismo encapsulado puede acomodar mucho circuitos, es mejor usar uno normalizado. El estándar que deben cumplir estos dados es el JESD51-4 [JED97]. En la siguiente figura se muestra una microfotografía de uno de estos circuitos, obtenida de [Del02]

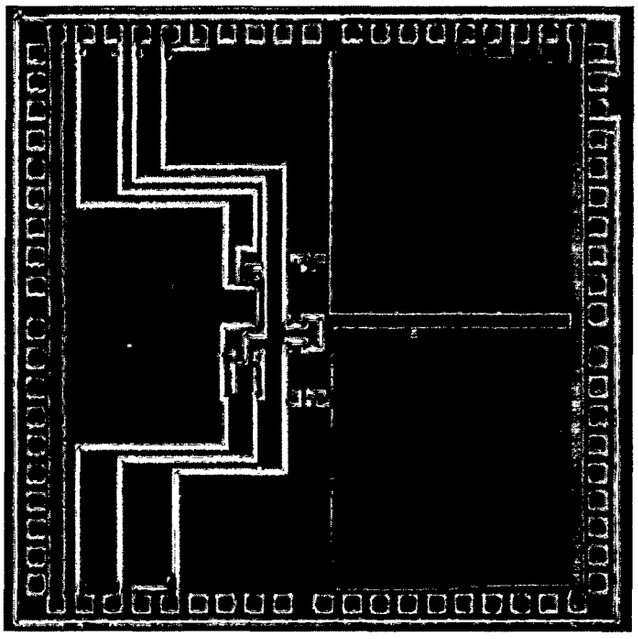


Fig. 7: Microfotografía de un dado para pruebas térmicas

Como es obvio, las desventajas de esta estrategia es que las medidas no son reales; ya se vio en el punto anterior que esto representa un problema, pues las resistencias térmicas dependen del tamaño del dado.

### *1.1.2. Obtención de $\theta_{JA}$ y $\Psi_{JT}$*

El primer paso es montar el dispositivo en una placa estándar (JESD51-3 [JED96] o JESD51-7 [JED99]), normalmente con dos (2L/0P) o cuatro capas (4L/2P), aunque puede haber otras combinaciones. En el caso de cuatro capas, como es habitual las dos internas se corresponden con planos de alimentación y masa. En la siguiente figura se muestra un ejemplo de estas placas, en particular la que se utiliza para caracterizar encapsulados PQFP desde 44 hasta 176 pines, obtenida del propio [JED99]:

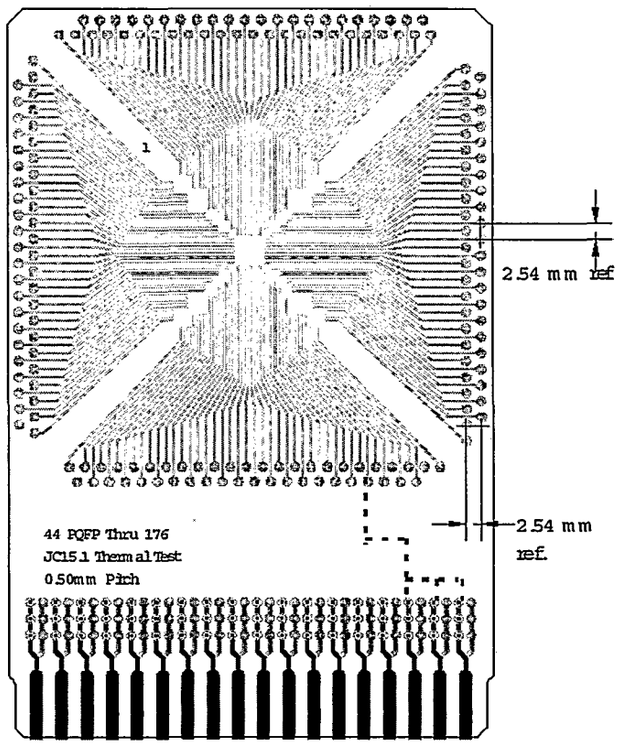



Fig. 8: Placa de pruebas estandarizada para medidas de resistencia térmica

A continuación se introduce la placa en un receptáculo estándar (JESD51-2 [JED95]) de un pie cúbico. Básicamente es una caja construida con materiales de baja conductividad térmica (madera o plásticos como policarbonato o polipropileno) y con un espesor mínimo de un octavo de pulgada (3,175 mm). El estándar también define la posición en la que debe quedar la placa y como fijarla, según se puede ver en la siguiente figura, tomada del estándar:

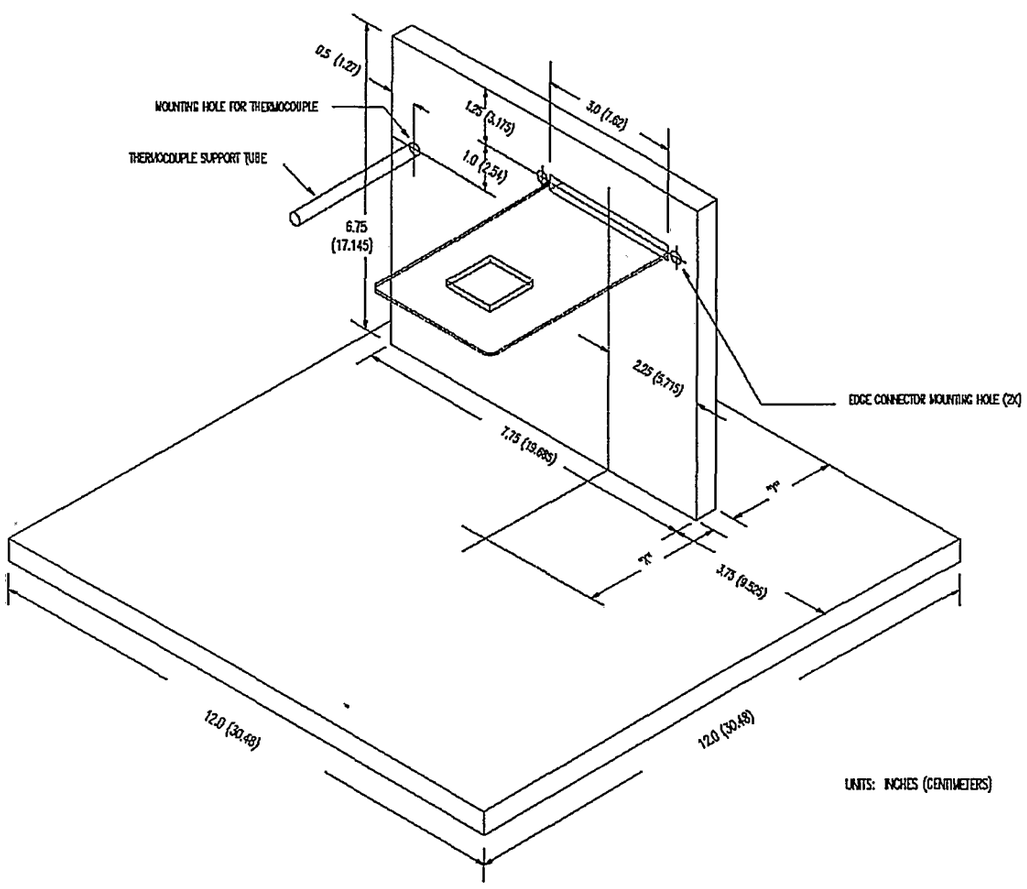



Fig. 9: Montaje estándar para medir $\theta_{JA}$ y $\Psi_{JT}$

Para poder empezar el experimento el ambiente debe estar entre 20 °C y 30 °C, y la temperatura dentro de la caja debe estar estabilizada (cambios de menos de 0.2 °C). En este momento se puede activar el circuito, y su consumo se fijará dependiendo de la $\theta_{JA}$ esperada. Por ejemplo, para entre 30 °C·W$^{-1}$ y 60 °C·W$^{-}$1, el estándar recomienda disipar 1 W. Una vez que se haya vuelto a alcanzar el equilibrio, se deberá medir la temperatura en el silicio, la potencia aplicada (V x I), la temperatura ambiental dentro del receptáculo (para calcular ) y la temperatura en el centro del encapsulado (para obtener $\Psi_{JT}$).

En el caso que se quieran obtener los valores cuando hay presente un flujo de aire, se emplea el estándar JESD51-6 [JED99b]. Para ello se utiliza un túnel de viento de tipo Eiffel (de succión, con el ventilador al final del recorrido), de baja velocidad (normalmente menor de 10 m/s) y baja turbulencia (2 %). El estándar define todos los parámetros del

experimento: las dimensiones de la cámara de pruebas, las especificaciones mínimas del túnel, la posición de los termopares, etc... Las placas que se utilizan son las mismas que para el caso de convección natural.

### *1.1.3. Obtención de $\theta_{JC}$*

El valor de se obtiene según los estándares SEMI G30 [SEM87] y G43 [SEM88], empleando el método de placa fría. Un método similar se describe también en MIL-STD 883, método 1012 [MIL96]. Esta técnica se basa en cubrir todo el encapsulado con una placa de cobre refrigerada con agua, según se muestra en la siguiente figura:

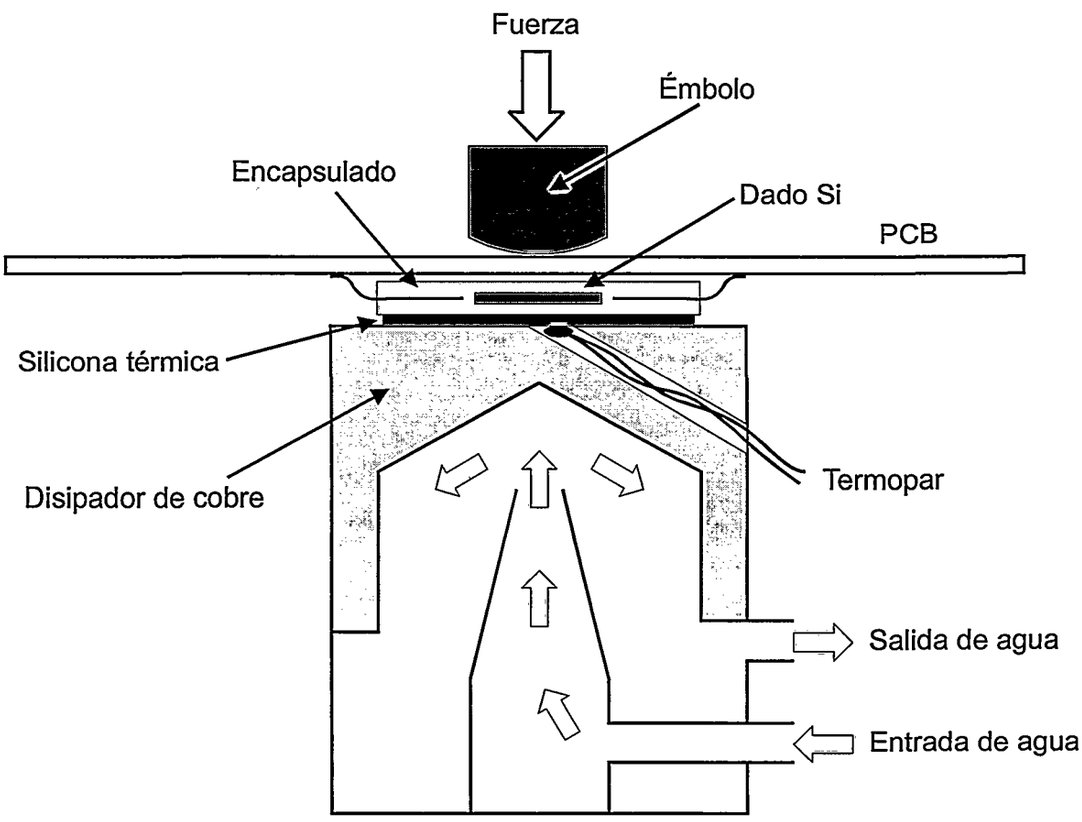



Fig. 10: Montaje estándar para medir $\theta_{JC}$

Así se consigue tener un disipador prácticamente ideal (con una resistencia térmica casi de 0 °C·W$^{-1}$). De esta manera se fuerza que todo el flujo de calor se escape a través

de la placa fría. Esto quizá quede más claro pensando en términos del modelo eléctrico equivalente: en una red eléctrica que tenga un camino de 0 ohmios, toda la corriente irá por esa rama. Y como el equivalente para la corriente es el flujo térmico, entonces todo el calor se disipará a través de la placa fría. Así lo que se consigue es hacer despreciables los flujos secundarios de calor del encapsulado, que podrían estropear la medida, y al mismo tiempo se uniformiza la temperatura en su superficie, pues la placa lo cubre en su totalidad.

Una vez montado el experimento, se activa el circuito, y cuando se alcanza el equilibrio se mide la potencia disipada, la temperatura en la unión y la temperatura en la superficie del encapsulado (a través de un termopar montado en la placa fría). Como ya se ha mencionado, el punto débil de $\theta_{JC}$ viene precisamente de como se obtiene. Por ello sólo puede considerarse como un dato válido cuando se utiliza en sistemas que sean similares al montaje que se acaba de describir: o sea, cuando se use un buen disipador y se tenga garantizado que la mayor parte del calor se disipa a través de él. Sólo en este caso puede ser un dato válido para calcular la temperatura de la unión; si no es así, probablemente sea más adecuado usar $\Psi_{JT}$.

## 2. Disipación de potencia

Es universalmente conocido que el incremento de temperatura en los circuitos integrados se produce por la potencia eléctrica que disipan, a través del efecto Joule. No por muy sabida esta afirmación deja de ser crucial: de ella se deduce que para poder predecir la temperatura de operación de un dispositivo es imprescindible que se pueda conocer de antemano su consumo.

El consumo en un circuito CMOS tiene tres componentes:

$$P = I_{fugas} V_{DD} + \sum_{nodos} Q_{SC} V_{DD} f_{nodo} + \frac{1}{2} \sum_{nodos} C_{nodo} V_{DD}^2 f_{nodo}$$

Donde $V_{DD}$ es la tensión de alimentación, $I_{perdidas}$ es la corriente total de fugas, $Q_{SC}$ es la carga de cortocircuito CMOS, $f_{nodo}$ es la frecuencia real de conmutación de cada nodo, y $C_{nodo}$ es la capacidad total de cada nodo del circuito.

La primera componente corresponde con la corriente de fugas de los transistores, y está causada por factores puramente tecnológicos (materiales, geometría y diseño de los

dispositivos,...). Si bien resulta un problema cada vez más grave según se va disminuyendo el tamaño de los transistores, por ahora es más o menos despreciable en tecnologías de producción.

El segundo termino modela la corriente de cortocircuito, es decir, la que se produce en el instante en que cambia la salida de una puerta, durante el cual sus transistores P-MOS y N-MOS estarán conduciendo simultáneamente. Para circuitos bien diseñados este término es también despreciable, pues se ajustan las geometrías de los transistores de tal manera que se minimice esta componente.

La última es componente es la realmente importante, pues en todos los circuitos digitales bien diseñados, y en particular en las FPGAS, puede llegar a suponer el 90% de la potencia dinámica (la que depende de $f_{nodo}$) [Poo02]. Se corresponde con la potencia disipada en cargar y descargar las capacidades parásitas de todos los nodos del circuito. Al igual que en la potencia de cortocircuito, depende directamente de la frecuencia real a la que conmuta cada nodo.

## 2.1. Evolución del consumo de potencia

Por supuesto, el primer punto que hay que tener en cuenta antes de seguir adelante es si la potencia (y la potencia) va a seguir siendo un problema en el futuro, o si por el contrario las nuevas tecnologías conseguirán reducirlo significativamente. Desafortunadamente para los diseñadores, la tendencia es que el consumo cada vez se vaya haciendo más grande. Como se puede ver en la Fig. 11, las predicciones de la SIA en el año 1999 ya indicaban que la potencia iba a crecer muy considerablemente en los siguientes años. Pero en la actualización del año 2000 las predicciones se corrigieron, para hacerlas todavía más pesimistas [SIA00].

La Fig. 12 [Tiw98] ilustra también muy bien el problema. La introducción de una nueva familia de microprocesadores significa un significativo incremento en el consumo. Este incremento se va solucionando en parte según el producto madura y se va migrando a tecnologías más modernas, hasta que llega un nuevo desarrollo otra vez con requerimientos mucho mayores de potencia.

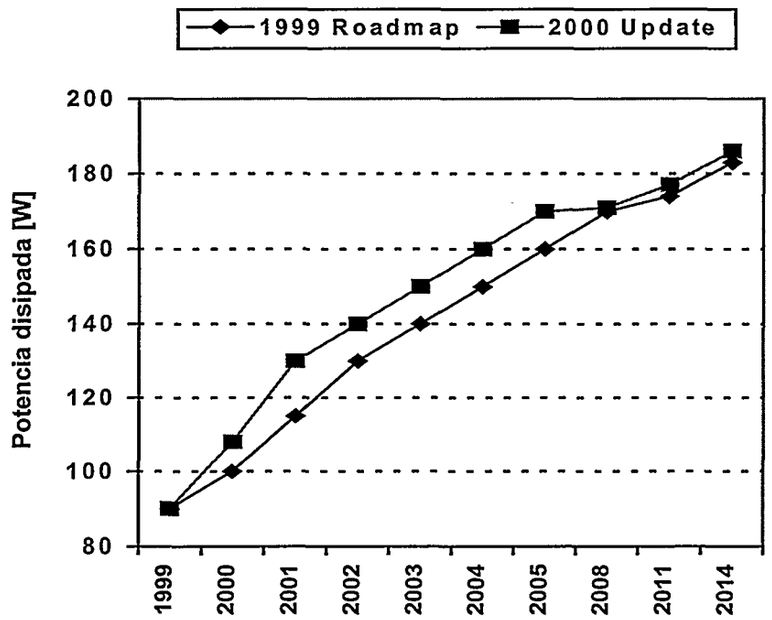



Fig. 11: Predicciones de 1999 y actualización de 2000 para el consumo de un C.I. de altas prestaciones

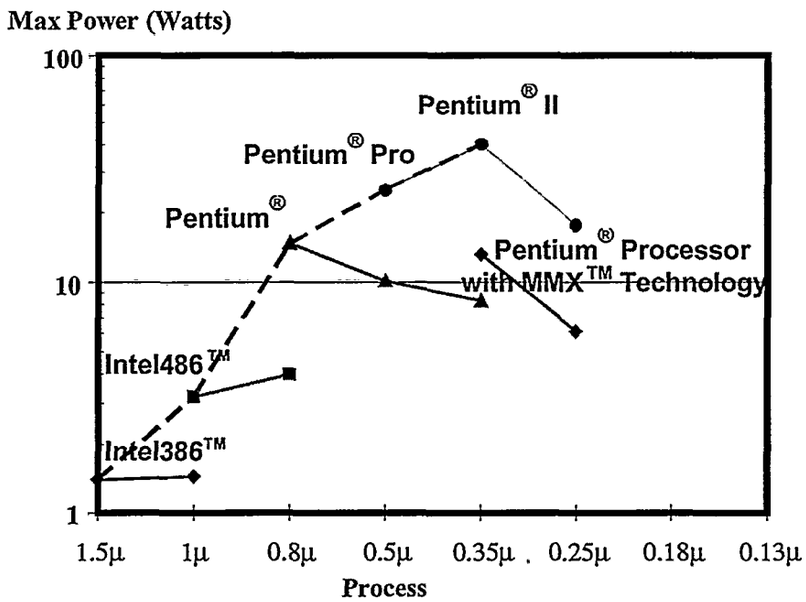



Fig. 12: Evolución en el consumo de los microprocesadores de Intel (obtenida de [Tiw98])

## 2.2. Estimación del consumo

Un punto imprescindible para poder predecir la temperatura de operación de un circuito integrado es ser capaz de estimar con precisión la potencia que disipa; esto es tan importante (o más) que tener un buen modelo del encapsulado. Y sin embargo, es una tarea muy compleja, pues como se ha visto depende de la frecuencia $f_{nodo}$ a la que conmute cada nodo. O sea, que la potencia dependerá de los datos que se estén procesando: por esta razón hay trabajos que tratan de modelar la potencia típica, otros que buscan averiguar la potencia máxima... Cada uno con sus ventajas y desventajas: el valor típico será el que ocurrirá la mayor parte del tiempo; pero 'típico' es inherentemente subjetivo. Por otro lado, el valor máximo será el más adecuado para dimensionar los disipadores, pero la mayor parte del tiempo se estará trabajando por debajo de este límite; puede ser incluso que se corresponda con una combinación de datos que no ocurre nunca en la realidad.

La estimación del consumo de FPGAs es sin duda un tema de investigación muy activo; se están desarrollando técnicas basadas en la propagación de probabilidades de conmutación [Poo02], o técnicas estadísticas basadas en simulaciones [Tod02], etc... Algunos artículos, ya clásicos, donde se compendian las posibles alternativas, más en el contexto de VLSI en general, son [Naj94, Ped97, Mac98].

Ya en el contexto de esta tesis, los dos principales fabricantes de FPGAs ofrecen herramientas de estimación de consumo: XPower de Xilinx, y de PowerGauge de Altera. Conceptualmente son muy sencillas; no son más que bases de datos con las capacidades parásitas de todos los nodos de los dispositivos. Para realizar la estimación se pueden usar los datos provenientes de una simulación, o usar valores típicos para la frecuencia de conmutación de los nodos. Obviamente la primera opción es la más recomendable, pero siempre que la simulación se corresponda con la realidad. Porque lo que estas herramientas no ofrecen es la generación automática de vectores de test; este es un tema hoy por hoy restringido al software de investigación. Y ese es probablemente su punto más débil, pues dependen de la pericia del diseñador a la hora de realizar una simulación que se corresponda fielmente con lo que ocurrirá en la realidad. Las pruebas realizadas en nuestro laboratorio sobre circuitos aritméticos no fueron muy positivas, llegándose a medir diferencias de hasta un 200% utilizando incluso los mismos vectores de pruebas [Sut02].

# 3. Fiabilidad

Desde hace mucho se tiempo se conoce que una elevada temperatura de operación de un circuito integrado acaba siendo perjudicial para su durabilidad a largo plazo. La justificación de este hecho ha venido dada históricamente por la ecuación de Arrhenius, que gobierna la totalidad de los procesos químicos, y muchos de los físicos:

$$R = A \cdot e^{-\frac{E_a}{KT}}$$

Donde R es la constante de ratio, A es una constante, $E_a$ es la energía de activación, K la constante de Boltzmann ($8{,}62 \cdot 10^{-5}$ eV/°K), y T la temperatura [°K].

La ecuación lo que hace es describir como los átomos o las moléculas alcanzan a través de su energía cinética (o sea, de su propia temperatura) la energía de activación. Esta energía es la que se necesita que pueda ocurrir un determinado fenómeno físico o químico. Cuanto mayor sea la temperatura, más posibilidades habrá de que se alcance la energía de activación, y por lo tanto subirá la constante de ratio, o sea, la rapidez con la que se va a producir el fenómeno. Algunos ejemplos de fenómenos regidos por esta ecuación podrían ser: una reacción química cualquiera, o un electrón que alcanza la banda de conducción en un semiconductor, o el movimiento un defecto en la red cristalina... Lo más importante es que todos estos ejemplos representan mecanismos de fallo en un chip, por lo que la conclusión es que los fallos de los circuitos integrados siguen la ecuación de Arrhenius. O lo que es lo mismo, que los incrementos de temperatura hacer aumentar exponencialmente la tasa de fallos de un circuito. Hay infinidad de referencias que respaldan esta teoría, un buen compendio es [JED01], otras referencias son

Pero en cualquier caso este modelo tiene un problema: que no ha podido ser validado experimentalmente. Hoy en día un valor de 100 años para el tiempo medio hasta un fallo (a $T_J$ = 25 °C) puede considerarse incluso hasta mediocre, con lo cual resulta imposible hacer experimentos en condiciones reales. Aunque hay muchas pruebas de que este modelo puede ser correcto, también es cierto que hay experimentos que dan resultados contradictorios. Un comentario muy ilustrativo puede ser el que pronunció Takehisa Okada, director senior de Sony Corporation [Lal96]:

*"We have a headache with Arrhenius"*

Una excelente referencia donde se comentan los problemas de este modelo es [Lal96], otros artículos que hablan del tema son [Jac97, Pec97].

Como conclusión, el modelo de Arrhenius proporciona un buen punto de partida para explicar la influencia de la temperatura en la fiabilidad, y en efecto es el estándar en la industria. Pero no debe considerarse que esta sea la única manera en la que la temperatura influye en la fiabilidad: existen gran cantidad de efectos de segundo orden que deben ser analizados para cada caso en particular.

## 3.1. Definiciones de términos

La unidad que se emplea para medir la tasa de fallos son los FITs (*failures in time*), que representa el número de fallos que se producen por cada millardo ($10^9$) de horas acumuladas de operación. En años, serían algo más de ciento diez mil.

Para calcular esta tasa de fallos se utiliza una muestra grande de chips, y tras un determinado tiempo de funcionamiento se comprueba cuantos han fallado. La proyección de la tasa de fallos obtenida para esta muestra en particular al caso general se hace a través de una distribución chi-cuadrado, según esta fórmula:

$$\lambda = \frac{\chi^2(2n+2,1-\alpha)}{2} \cdot \frac{10^9}{ss \cdot t \cdot AF}$$

donde $\lambda$ es la tasa de fallos [FITs], $\chi^2(2n+2,1-\alpha)$ es el valor de chi-cuadrado para 2n+2 grados de libertad y probabilidad 1-$\alpha$; n es el número de chips que han fallado y $\alpha$ es el grado de confianza. Por último, ss es el tamaño de la muestra, t el tiempo del experimento (en horas) y AF el factor de aceleración (descrito en el siguiente punto). En la siguiente tabla se muestran algunos valores de chi-cuadrado:

| $\chi^2/2$ | 60% confianza $\alpha = 0,6$ | 90% confianza $\alpha = 0,9$ | 95% confianza $\alpha = 0,95$ |
|---|---|---|---|
| n = 0 | 0,91629008 | 2,30258806 | 2,99573818 |
| n = 1 | 2,022313423 | 3,88971698 | 4,74386423 |
| n = 2 | 3,105378385 | 5,32231874 | 6,29578871 |

Tabla 2: Algunos valores comunes de $\chi^2/2$

Así, si se tiene una muestra de 100 dispositivos y falla uno (n=1), se podrá decir con un 60% de confianza ($\alpha$=0,6) que la tasa de fallos es 2,02% (el valor de $\chi^2/2$ entre el tamaño de la muestra). Si subimos la confianza al 95%, la tasa sube hasta el 4,74%.

La distribución que siguen los fallos en el tiempo se suele considerar que es exponencial pura, lo que implica una tasa de fallos constante en el tiempo:

$$F(t) = 1 - e^{-\frac{t}{c}}$$

Donde c representa la vida media del circuito. Como se justifica en [JED01], esta es una buena aproximación para experimentos con muestras grandes y pocos errores, y además, puesto que los fallos en los circuitos integrados son procesos tan lentos, normalmente no hay suficientes datos experimentales como para justificar que haya una distribución que se ajuste mejor. Además, la distribución exponencial tiene una ventaja adicional, y es que permite calcular el MTTF (tiempo medio hasta un fallo) de una manera muy sencilla:

$$MTTF = \frac{10^9}{\lambda}$$

Donde el MTTF se expresaría en horas, aunque para que sea más manejable se suele pasar a años.

Aunque esta distribución sea la que se usa comúnmente, por lo sencilla que es de usar, hay referencias que indican [NIS02] que en la realidad la distribución que mejor se aproxima es la de Weibull:

$$F(t) = 1 - e^{-\frac{t}{c}m}$$

Donde la tasa de fallos no es constante en el tiempo, sino que varía según un factor de forma m. Este valor es mayor que 1 (tasa de fallos mayor) al comienzo y al final de la vida del dispositivo. El incremento al principio se corresponde con el fallo los dispositivos con errores de fabricación, y el del final, por la llegada al término de la vida del circuito. En la Fig. 13 se muestra la evolución de la tasa de fallos; esta gráfica se la suele denominar como la "curva de la bañera" ("*bathtub curve*").

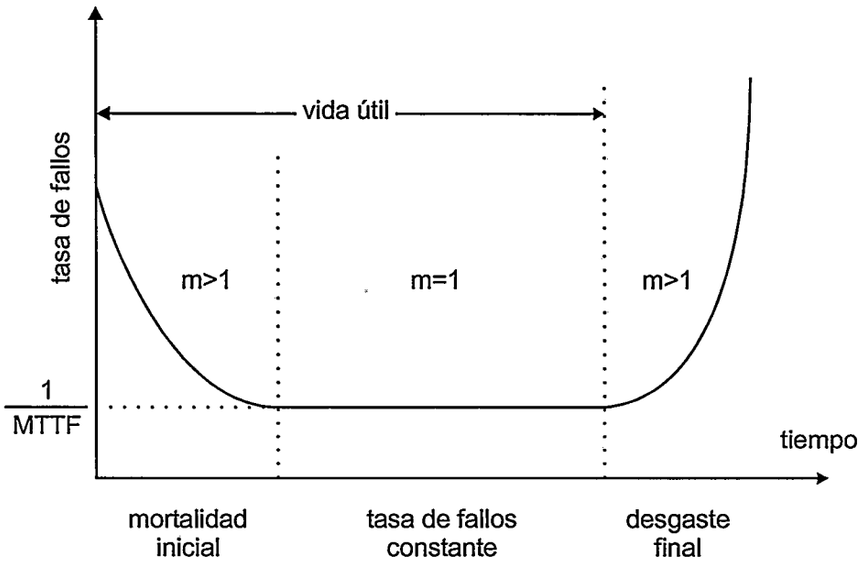



Fig. 13: Evolución típica de la tasa de fallos de un circuito integrado

Sin embargo, y aunque este modelo pueda parecer más adecuado, lo cierto es que es más difícil de manejar que el de la distribución exponencial, y además, la longevidad de los circuitos integrados hace muy difícil calcular la variación en el tiempo de la tasa de fallos. Es por esto que no tiene mucha utilidad en la práctica, donde se usa siempre para hacer los cálculos las distribución exponencial.

A continuación se presenta una tabla con las tasas de fallos y el MTTF para una serie de familias de FPGAs [Xil02b], entre ellas dos de las usadas en esta tesis:

| | FITs | | MTTF (años) | |
|---|---|---|---|---|
| | $T_j$ = 25 °C | $T_j$ = 55 °C | $T_j$ = 25 °C | $T_j$ = 55 °C |
| XC4000E | 2 | 25 | 57077 | 4566 |
| XC4000XL | 2 | 30 | 57077 | 3805 |
| XCS (Spartan) | 1 | 16 | 114155 | 7134 |
| XCV (Virtex) | 1 | 10 | 114155 | 11415 |
| XCVE (Virtex-E) | 2 | 23 | 114155 | 4963 |

Tabla 3: Tasas de fallos y MTTF para familias de FPGAs comúnmente usadas

Como se puede ver, los tiempos hasta el fallo son artificiosamente longevos; esto es así porque se obtienen como el inverso de la tasa de fallos, que trabaja con intervalos de tiempo extraordinariamente largos. En cualquier caso se puede comprobar como un incremento moderado de la temperatura (30 °C) provoca una disminución de un orden de magnitud en la fiabilidad.

## 3.2. Obtención de las tasas de fallo de un circuito integrado

Como ya se ha visto, las tasas de fallo de los circuitos integrados son afortunadamente muy bajas. Pero esto implica un problema, y es que con esta longevidad se tardarían siglos en hacer los experimentos. Por lo tanto es necesario utilizar algún método de envejecimiento acelerado. Y si se da por válido el modelo de que los fallos en los circuitos integrados se activan térmicamente, entonces el método a seguir es muy sencillo: sólo hay que probar los chips a temperaturas muy elevadas. Puesto que en la ecuación de Arrhenius la dependencia de la constante de ratio con la temperatura es exponencial, es de esperar que al hacer las pruebas a altas temperaturas la tasa de fallos será grande, de tal manera que se podrá medir con una precisión adecuada en un tiempo razonable. Y luego este valor se extrapola para obtener la tasa de fallos a la temperatura normal de funcionamiento.

Este método es en efecto el estándar para calcular las tasas de fallos, tal y como recomienda JEDEC en [JED01]. En esta publicación se define el factor de aceleración:

$$AF = e^{-\frac{E_a}{k}\left(\frac{1}{T_t}-\frac{1}{T_s}\right)}$$

Que sirve para calcular a lo que equivale el tiempo de pruebas a temperatura de stress ($T_t$) en horas de funcionamiento a la temperatura normal de operación ($T_s$) [Alt02b]:

$$\textit{Horas equivalentes a temperatura normal} = \textit{Horas en stress} \times AF$$

Y así, la tasa de fallos a temperatura normal se calcula con la siguiente ecuación, ya presentada al comienzo de este punto [JED01, Xil02b, Alt02b]:

$$\lambda = \frac{\chi^2(2n+2,1-\alpha)}{2} \cdot \frac{10^9}{ss \cdot t \cdot AF}$$

Aunque en realidad esta es una versión simplificada de la correcta [HMP99]

$$\lambda = \sum_{i=1}^{\beta} \left( \frac{n_i}{\sum_{j=1}^{k} ss_j \cdot t_j \cdot AF_{ij}} \right) \cdot \frac{\chi^2 (2\sum_{i=1}^{\beta} n_i + 2{,}1 - \alpha) \cdot 10^9}{2 \cdot \sum_{i=1}^{\beta} n_i}$$

Donde $\beta$ es el número total de posibles mecanismos de fallo, $n_i$ es el número de circuitos que han fallado por el mecanismo i=1,2,... $\beta$, k es el número de test que se realizan, $t_j$ es la duración de cada test j=1,2,...k, $ss_j$ es el tamaño de muestra para cada test j=1,2,...k

Esta ecuación tiene en cuenta que los fallos se pueden producir por distintos mecanismos, cada uno con una energía de activación distinta, y por tanto, con un factor de aceleración diferente. Esto implica que para hacer un cálculo riguroso de la tasa de fallos, hay que hacer un análisis post-mortem de los chips para averiguar las razones de su fallo, para usar la correcta $E_a$. Si no se llega a conocer el origen del fallo, se toma una $E_a$ por defecto de 0,7 eV, y para los fallos que no se ha podido averiguar experimentalmente su $E_a$, se toma un valor de 1,0 eV.

En la realidad el proceso de test es todavía un poco más complejo, pues el stress no sólo se hace a temperaturas elevadas, sino también a tensiones más altas de las normales, o humedad relativa más elevada, o con ciclos de temperatura... todo esto hace que el factor de aceleración para cada fallo en particular tenga una componente que siga el modelo de Arrhenius, y otra componente que siga otro modelo, según se detalla en [JED01]. En cualquier caso estos modelos adicionales son sencillos: una potencia o una exponencial. Algunos ejemplos podrían ser [Alt02b] la aceleración para rupturas en el óxido de la puerta de los transistores:

$$AF = e^{\frac{\gamma}{t_{ox} \cdot 10nm}(V_{stress} - V_{operation})}$$

O en el dieléctrico entre capas:

$$AF = e^{\gamma \cdot (V_{stress} - V_{operation})}$$

Donde $\gamma$ es un factor que dependerá de cada mecanismo de fallo.

A modo de resumen, en la siguiente tabla se detallan algunos mecanismos de fallo junto con sus correspondientes energías de activación [Alt02b]:

| Mecanismo | Energía de Activación $E_a$ [eV] | Factor Exponente de Voltaje ($\gamma$) |
|---|---|---|
| Rotura del óxido de puerta | 0,7 | 3,2 |
| Defecto entre capas | 0,7 | 2,0 |
| Errores en las máscaras | 0,5 | 0 |
| Defecto en la metalización | 0,5 | 0 |
| Electromigración | 0,7 (Al-Si), 1,0 (Al-Cu) | Depende de la densidad de corriente |
| Contaminación | 1,0 | 0 |
| Pérdida de carga | 0,6 | 0 |
| Electrones calientes | -0,17 | Depende de la corriente de sustrato |

Tabla 4: Energías de activación y factores de voltaje para diversos mecanismos de fallo

### *3.2.1. Determinación de las energías de activación de los fallos*

Los valores de las energías de activación para los diferentes mecanismos de fallos no son ni mucho menos arbitrarios: se obtienen mediante experimentación. Para ello, se deben hacer al menos dos pruebas a diferentes temperaturas, que proporcionarán diferentes valores del tiempo hasta el fallo. Por supuesto, deben escogerse los fallos producidos por el mismo mecanismo (una vez más, un análisis post-mortem es necesario). Estos valores deben cumplir que:

$$\ln(t_{fallo1}) = C + \frac{E_a}{kT_1} \qquad \ln(t_{fallo2}) = C + \frac{E_a}{kT_2}$$

O sea, que la gráfica del logaritmo del tiempo hasta el fallo versus el inverso de la temperatura debe producir una línea recta. Si no es así, es que el mecanismo de fallo no está activado por la temperatura. Restando ambas ecuaciones y despejando $E_a$:

$$E_a = k \cdot \frac{\ln(t_{fallo1}) - \ln(t_{fallo2})}{\left(1/T_1 - 1/T_2\right)}$$

# 4. Soluciones térmicas

En este punto se tratará de introducir brevemente las soluciones que hay para disipar el calor de un circuito integrado, que básicamente son cinco [Int98]. La más sencilla es usar un disipador pasivo, que utilice convección natural (sin flujo de aire forzado). Con esta solución se pueden disipar entre 5 y 25 W aproximadamente. Si se necesita disipar algo más de potencia (entre 15 y 50 W) se pueden recurrir a una solución semi-activa, que utiliza el flujo de aire de los ventiladores del sistema. Más calor todavía, entre 10 y 160 W, se puede disipar utilizando un ventilador dedicado para cada circuito integrado. Y para manejar una cantidad de calor prácticamente ilimitada, siempre se puede usar refrigeración líquida. Por último, si no se puede poner el disipador directamente sobre el chip, se puede transferir usando tuberías de calor, que pueden manejar típicamente entre 100 y 150 W.

## 4.1. Disipadores

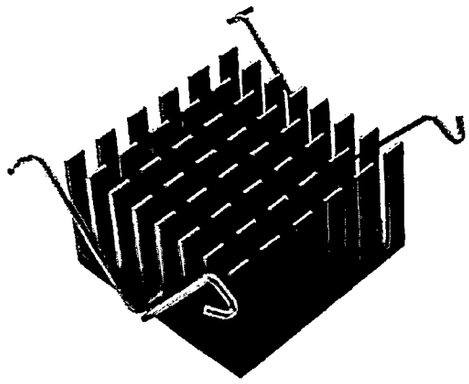
Algo que todos los ingenieros saben es que los disipadores se basan en un principio muy simple: aumentar el área efectiva por la que se puede disipar el calor. Cuanto mayor sea el área, mayor será esta disipación, y menor será la resistencia térmica. Menos conocida es la relación entre ambas magnitudes, que puede expresarse de una manera aproximada como [Int98]:

$$\theta_d = \frac{50}{\sqrt{A}}$$

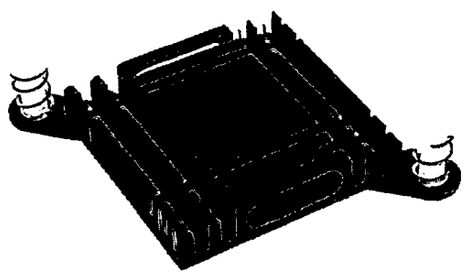
Donde A es el área del disipador [$cm^2$] y $\theta_d$ es la resistencia térmica del disipador [$°C \cdot W^{-1}$]. Como ya se vio en el punto 1, la resistencia térmica desde la unión hasta el ambiente $\theta_{JA}$ puede obtenerse como la combinación:

$$\theta_{JA} = \theta_{JC} + \theta_d$$

En la siguiente figura se muestran dos de los posibles disipadores que se podrían utilizar con un encapsulado muy común, el FG456. Se observa la influencia que tiene la geometría en los valores de $\theta_d$, y que estos valores son muy pobres a no ser que no se emplee ventilación forzada. Esto es debido al pequeño tamaño de las FPGAs (23 x 23 *mm para este ejemplo), que impide que los disipadores puedan tener un área respetable.*



| Altura [mm] | $\theta_d$ (0 LFM) | $\theta_d$ (200 LFM) |
|---|---|---|
| 10.0 | 40.0 | 11.69 |
| 18.0 | 23.4 | 7.39 |
| 25.0 | 19.7 | 6.37 |



| Altura [mm] | $\theta_d$ (0 LFM) | $\theta_d$ (200 LFM) |
|---|---|---|
| 6.0 | 44.1 | 13.13 |
| 10.0 | 30.6 | 9.26 |

Fig. 14: Disipadores que se pueden utilizar con el encapsulado FG456 [Aav02]

Estas cifras prácticamente impiden usar convección natural; esto es algo muy común en los disipadores tan pequeños, muchos fabricantes sólo ofrecen el valor de $\theta_d$ con ventilación forzada. Aunque la solución más sencilla es poner un ventilador directamente sobre el disipador, otra solución válida (y probablemente más barata) es utilizar uno o varios ventiladores que proporcionen un flujo de aire a todo el PCB. No se consiguen tan buenos resultados como con un ventilador dedicado, pero en cualquier caso la capacidad de disipar calor se ve considerablemente incrementada, llegando incluso hasta triplicarse con velocidades de aire muy moderadas, de sólo 1 $m \cdot s^{-1}$. El mayor problema de esta solución viene de lo complejo que es realizar los cálculos. Como se verá en el punto siguiente, puede ser necesario incluso recurrir a métodos de dinámica de fluidos. Sin embargo, en el caso de utilizar un ventilador integrado en el disipador, estos cálculos ya están hechos por el fabricante, por lo que no hay que preocuparse: el valor que se da para $\theta_d$ ya considera el efecto del ventilador.

## 4.2. Ventiladores

El flujo de aire necesario para disipar una determinada de calor es muy sencillo de calcular utilizando esta fórmula [Tur96]:

$$G = \frac{Q}{\rho \cdot C_p \cdot \Delta T}$$

Donde G es el flujo de aire volumétrico [$m^3 \cdot s^{-1}$], Q es la cantidad de calor que se debe disipar [W], $\rho$ es la densidad del aire [$kg \cdot m^{-3}$], $C_p$ es el calor específico del aire [$J \cdot kg^{-1} \cdot °K^{-1}$] y $\Delta T$ es el incremento de temperatura [°K].

El problema viene de cómo funcionan los ventiladores: el flujo de aire que son capaces de dar no es fijo, sino que depende de la diferencia de presión la entrada y la salida del aire. La curva que fija esta dependencia es el parámetro fundamental para caracterizar un ventilador. Por otro lado, el sistema que se está ventilando impondrá una cierta resistencia al paso del aire, que se traducirá en un incremento de la presión según vaya aumentando el flujo de aire. Donde se crucen ambas curvas, será el punto de funcionamiento del ventilador, el que nos indicará cuál es el flujo de aire en cada caso en particular. Un ejemplo puede verse en esta figura:

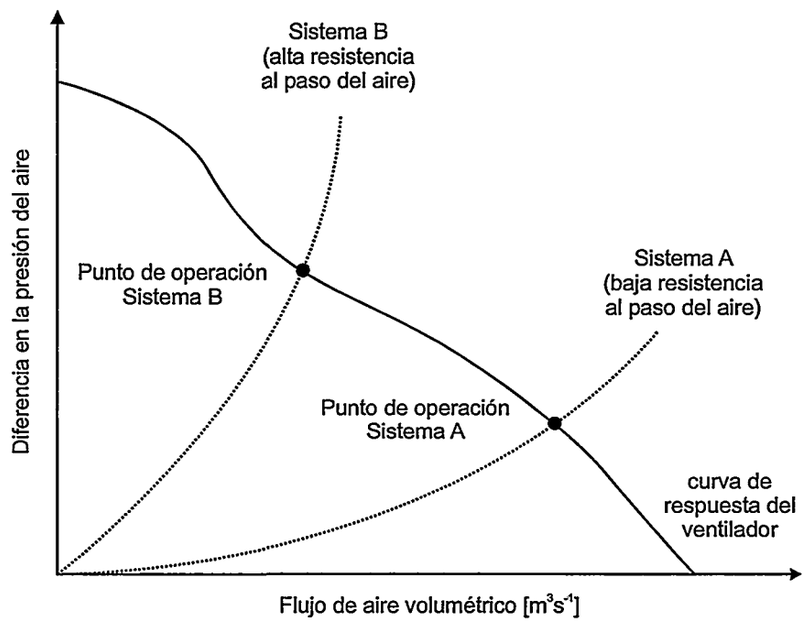



Fig. 15: Gráfica para calcular el flujo de aire aportado por un ventilador

Como se puede sospechar, calcular el punto de trabajo no es ni mucho menos sencillo. Para ello se puede utilizar alguno de estos tres métodos:

- Medida experimental del flujo de aire
- Utilizar un modelo similar al de un circuito eléctrico, como se describe en [Ell95]
- Emplear un software de dinámica de fluidos (CFD, *computational fluid dynamics*)

Sin duda alguna la última opción es la más adecuada, y la que proporciona los resultados más precisos, pero su gran inconveniente es lo complejo que resultan de hacer estos modelos, y que no son ni mucho menos herramientas sencillas de utilizar.

## 4.3. Tuberías de calor

En muchas ocasiones no hay espacio suficiente para instalar el disipador justo encima del circuito integrado. Esto es especialmente cierto en los sistemas portátiles, donde el tamaño es una grave limitación. Por ejemplo, si en los ordenadores portátiles se tuviera que poner el disipador (con su correspondiente ventilador) justo encima de la CPU (que habitualmente se sitúa debajo del teclado), entonces su grosor sería como mínimo el doble del que tienen en la actualidad, convirtiéndolos en cualquier cosa menos en portátiles.

En estos casos es necesario contar con un mecanismo que permita transportar el calor de una manera muy eficiente, desde el circuito de calor que lo genera hasta el disipador que lo libera. Esta función la realizan las tuberías de calor (*heatpipes*), que son elementos con una conductividad térmica excepcional: entre 10 y 10.000 veces mejor que la del cobre. Como se muestra en la Fig. 16, estos elementos son básicamente tubos cerrados cuyas paredes están recubiertas de un material poroso, y en cuyo interior hay agua u otro líquido (esto dependerá de la temperatura de operación). El tubo se calienta por un lado, y el líquido se transforma en vapor, que viaja por el centro hasta que llega al extremo frío, donde se condensa y vuelve por capilaridad y/o gravedad a través de la estructura porosa (también llamada mecha) hasta el extremo caliente, donde se vuelve a repetir el ciclo.

En la actualidad estos elementos son muy empleados: prácticamente todos los ordenadores portátiles los utilizan. Incluso empiezan a aparecer en sistemas de sobremesa compactos, y hasta en tarjetas gráficas, como se ve en la Fig. 17.

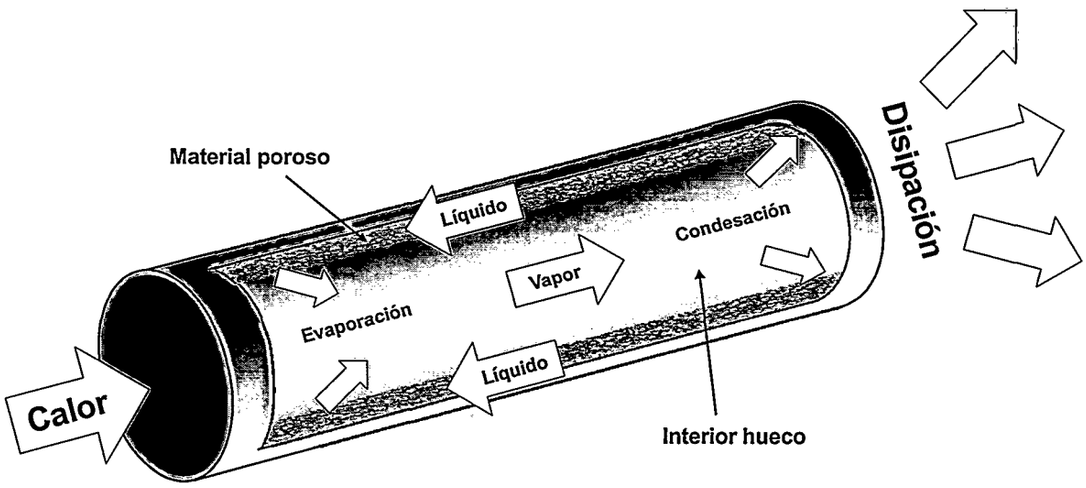



Fig. 16: Esquema de una tubería de calor (*heatpipe*)

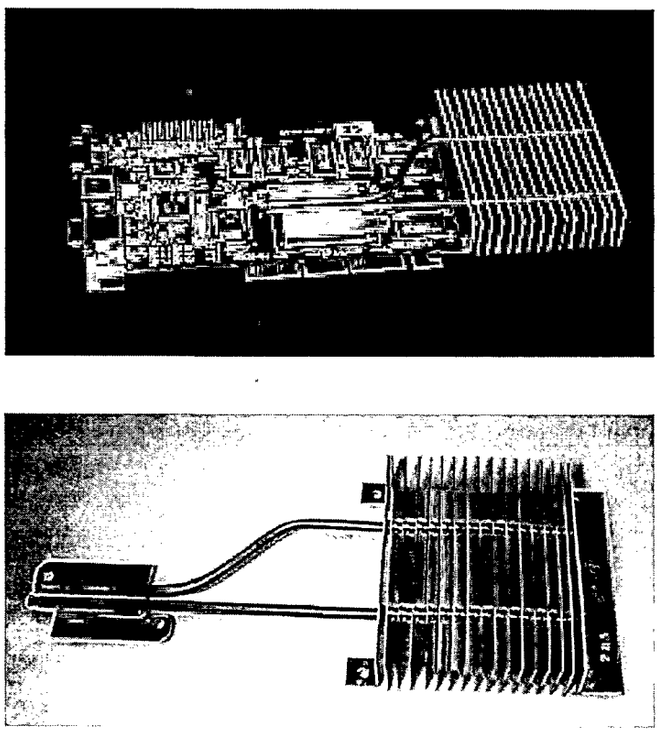


Fig. 17: Refrigeración de una tarjeta gráfica con tuberías de calor (obtenidas de [The99])

# 5. Protección térmica de los circuitos integrados

Las soluciones térmicas que se han descrito en el punto anterior son muy variadas, pero en cualquier caso tienen un punto común: ninguna de ellas puede considerarse libre de fallos. Se puede soltar el clip de sujeción de un disipador pasivo, o puede ser que un ventilador no llegue a arrancar por que esté lleno de polvo, o que se corroa una tubería de calor y pierda su líquido interno (y con él, su capacidad para transportar el calor), o que ocurra cualquier otro desastre... Todos estos fallos provocan un incremento repentino de la resistencia térmica $\theta_{JA}$ que ve el circuito integrado, y si no se produce un descenso inmediato de la potencia disipada, la temperatura de la unión sube hasta que el chip deja de funcionar correctamente (por el incremento de los retardos), o en el peor de los casos, se autodestruye. Esto no es ni mucho menos exagerado; en el informe [Pab01] se muestra como al retirar el disipador de un AMD AthlonMP a 1,2 GHz los resultados son desastrosos: la temperatura sube a más de 300 °C. Aunque la placa base sobre la que se hizo esta prueba incluía un circuito para monitorizar la temperatura de la CPU (usando un diodo embebido), por un diseño cuando menos discutible sólo era capaz de detectar variaciones de 1 $°C \cdot s^{-1}$, de tal manera que cuando se revelaba el fallo, el procesador ya estaba destruido.

La solución para evitar que un fallo en el disipador se traslade al circuito integrado es implementar un mecanismo de protección; este mecanismo estará compuesto por un sensor de temperatura integrado, que cuando detecta un sobrecalentamiento reduce la frecuencia del reloj del sistema. Esta reducción de la velocidad tiene una doble utilidad: por un lado, conseguir que el sistema siga funcionando aunque los retardos se incrementen por el calentamiento; y por otro, reducir la potencia consumida para evitar la autodestrucción del chip. Un buen ejemplo de esta estrategia es la protección incluida en el Pentium4 de Intel [Int01]: si la temperatura sube por encima un cierto límite, el reloj se modula, de tal manera que está activo sólo el 50% del tiempo. Si sigue subiendo y llega a 135 °C, el procesador se para completamente. En otros fabricantes, como AMD, este circuito de protección debe añadirse externamente [AMD03], utilizando las medidas tomadas de un diodo embebido en la CPU. Como se ha visto al comienzo de este punto, la desventaja de esta alternativa es que no es *stand-alone*, se depende de un circuito que si no está bien diseñado, no va a ser capaz de proteger el procesador frente a

incrementos súbitos de la temperatura. Por último, otra referencia de un fabricante de microprocesadores en la que se comenta este problema es [Tra02].

Un tipo de circuitos que disponen inherentemente de esta protección son los *self-timed* o autotemporizados [Sut89]. Estos circuitos están diseñados de tal manera que su velocidad de operación se adapta en cada momento a los retardos de sus distintos bloques. De esta manera, se produce una realimentación que protege el circuito: si la temperatura sube, el rendimiento (y por tanto, el consumo) baja, evitando así que la temperatura suba indefinidamente.

# 6. Conclusiones

La temperatura ha sido siempre, y probablemente lo sea ahora más que nunca, un factor muy importante en el funcionamiento de los circuitos integrados

- En primer lugar, una alta temperatura de operación causa un incremento de los retardos, y por lo tanto, una merma en las prestaciones
- Pero lo más importante es que la temperatura interviene muy negativamente en la fiabilidad de los circuitos integrados. En este capítulo se han presentado los modelos estándares para el cálculo de la tasa de fallos. Estos modelos se basan en la ecuación de Arrhenius, que tiene una dependencia exponencial con respecto a la temperatura.

Aunque conocer la temperatura del silicio resulta indispensable para optimizar el rendimiento y la fiabilidad, no es una tarea sencilla de realizar:

- Como se ha visto al comienzo de este capítulo, los clásicos modelos de resistencia térmica que se usan para caracterizar los encapsulados son muy pobres, y de poco sirven para predecir con precisión la temperatura de la unión
- También se ha indicado que modelar las distintas soluciones térmicas que se emplean habitualmente (disipadores, ventiladores,...) es todavía más difícil; necesitando de complejos análisis de elementos finitos.
- Además, en este capítulo también se ha presentado lo complejo que es predecir el consumo de potencia; si esto se une a lo expuesto en los dos

puntos anteriores, la conclusión es que no es posible predecir la temperatura del silicio con los datos que proporcionan los fabricantes de los circuitos integrados.

El corolario de todos estos puntos es que la única manera de conocer con precisión la temperatura de operación de un circuito integrado es embebiéndole un sensor en el silicio.

# Capítulo 3.

# Principales técnicas de medida de temperatura

En este capítulo se presentan los principales tipos de sensores que se han venido empleando en el estudio la temperatura de los circuitos integrados. En primer lugar se introducen cuales son los principales parámetros que describen un sensor, para luego pasar a hacer una descripción detallada de los cuatro tipos más usados. Por último, se hace una introducción de otras técnicas que por su complejidad o por otras limitaciones no son tan comunes, pero que en cualquier caso son bastante referenciadas en la literatura. Se ha tratado de hacer una descripción genérica, pero siempre teniendo en cuenta que los dispositivos de interés en esta tesis son las FPGAs.

## 1. Principales parámetros de un sensor de temperatura

### 1.1. Error

El error es un parámetro básico a la hora de evaluar un sensor. Al igual que pasa con cualquier otra magnitud física, podrá venir expresado de forma absoluta (± 2°C) o de forma relativa (± 5%). El error no debe considerarse responsabilidad únicamente del sensor, porque puede ocurrir en cualquier punto del proceso: en el circuito que se utiliza para manejarlo, o en la interpretación de los resultados... Por esta razón, para poder estimarlo se deberá hacer un análisis de todos los elementos que intervienen en la medida de la temperatura.

El error, a parte de por el ruido, esta principalmente causado por estos factores:

- Linealidad: es un error en la interpretación de los resultados, viene de la simplificación de que la respuesta del sensor es lineal, cuando en realidad se corresponderá con otra función.
- Calibración: el error viene de otra simplificación, la de considerar todos los sensores como si fueran iguales, obviando la dispersión en sus respuestas.
- Sensibilidad: si es baja puede facilitar el error por ruido. Por otro lado, también es causa de error que el transductor sea sensible a otros factores externos, que no sean la temperatura, pues cualquier variación en ellos producirá una alteración de la medida.

Todos estos parámetros se definen detalladamente en los siguientes puntos.

## 1.2. Necesidad de calibración previa

Dependiendo del error que se esté dispuesto a tolerar y de las propias características del sensor, puede ser que sea necesario realizar un calibrado previo en un horno con temperatura controlada. Por supuesto, lo más recomendable será utilizar transductores que no necesiten este calibrado. Pero si hay que hacerlo, esto no implica que tengamos que calibrar todos los dispositivos antes de montarlos, lo más habitual es que se produzcan escenarios más favorables:

- Que todos los dispositivos de un mismo tipo se comporten igual, o incluso,
- Que todos los dispositivos de una misma familia tengan la misma respuesta

En cualquier caso, será la estadística la que fije la estrategia a seguir, teniendo en cuenta siempre el error que se está dispuesto a tolerar. Una cuestión importante es que puede ser que el calibrado no sea igual de necesario en todos los parámetros de la respuesta del sensor. Así, si se emplea un sensor con una respuesta lineal, del tipo:

$$y = a + b \cdot T$$

Puede ser que sea necesario calibrar a, porque tiene mucha dispersión, pero sin embargo no haga falta para b porque es prácticamente igual para todos los sensores. En efecto, y como se verá más adelante, en los sensores que se han empleado en esta tesis esto es lo que ocurre. El parámetro b, que representa la sensibilidad del sensor frente a

los cambios de temperatura, varía muy poco de dispositivo a dispositivo, incluso para FPGAs de distinto tamaño pero de la misma familia. Esto es debido a que este factor está muy relacionado con la tecnología y no con las variaciones en el proceso de fabricación. Sin embargo, en los experimentos que se han realizado si que se ha observado que el parámetro a varía mucho de dispositivo a dispositivo, o incluso al variar la posición del sensor dentro de un mismo circuito.

En estos casos en los que las variaciones se dan sólo en un coeficiente de la respuesta del sensor, con calibrar únicamente en un punto queda resuelto el problema. Aparece entonces la distinción entre normalización y calibración:

- Normalización será cuando se haga un ajuste de las respuestas particulares de cada sensor tomando sólo una medida. Este es el caso que ocurrirá cuando sólo haya dispersión en uno de los coeficientes de la respuesta.
- Y por otro lado calibración será cuando se obtenga el comportamiento del sensor midiendo su respuesta para distintas temperaturas. Este es el caso general, cuando la dispersión afecte a todos los coeficientes de la respuesta.

La normalización es un proceso muy rápido, sólo hay que tomar una medida de temperatura, y por lo tanto no puede considerarse un grave inconveniente que un sensor la requiera. Sin embargo, la calibración es un proceso mucho más costoso en tiempo, pues cada medida puede necesitar varias decenas de minutos (por las abultadas constantes de tiempo de un horno con temperatura controlada). Por lo tanto, sólo serán útiles aquellos sensores en los cuales la calibración pueda ser extrapolada a todos los dispositivos del mismo tipo, o mejor todavía, de la misma familia, de tal manera que sólo haga falta hacer como mucho una normalización previa.

## 1.3. Sensibilidad

La sensibilidad se define como la variación de la respuesta del sensor frente a los incrementos de temperatura:

$$\frac{\Delta y}{y} = S\frac{\Delta T}{T}$$

Donde y es la salida del sensor, T es la temperatura y S la sensibilidad. Obviamente, cuanto mayor sea la sensibilidad más fácil será de usar el sensor. La mayor dificultad que

tienen los sensores con baja sensibilidad es que el ruido es mucho más crítico, porque habrá que amplificar mucho la señal. Y por esta razón, los circuitos para realizar el acondicionamiento de señal serán también más complejos.

## 1.4. Sensibilidad frente a otros factores externos

El sensor ideal es aquel que es sensible sólo a los cambios de temperatura del punto que se desea medir, de tal manera que ningún otro factor ambiental o de fabricación afecta a la medida. Obviamente este es un requerimiento utópico, por lo que puede ser que sea necesario establecer algún mecanismo de compensación, dependiendo siempre del error que se esté dispuesto a tolerar. Los factores más habituales a los que puede ser sensible un sensor son:

- Tensión de alimentación: se resuelve midiéndola directamente, o usando otro sensor con distinta sensibilidad frente a Vcc, y se obtiene indirectamente resolviendo una ecuación.
- Variaciones en el proceso de fabricación: se resuelve calibrando (o en el mejor de los casos, normalizando) la respuesta del sensor para cada dispositivo.
- Temperatura ambiente: se resuelve midiéndola con un sensor dedicado, normalmente mucho más sencillo de resolver que la medida en el silicio.

## 1.5. Linealidad

La respuesta ideal de un sensor es la lineal, sencillamente porque es la más sencilla de manejar. Por esta razón se buscan siempre transductores que tengan más o menos esta respuesta, aunque en la realidad nunca vaya a existir uno que sea completamente lineal. El resultado que se producirá es un error en la medida, pues se está suponiendo una respuesta que no es la real del sensor.

Algunas de las no linealidades más típicas son la saturación, las zonas muertas (puntos en el que el sensor tiene sensibilidad cero) y la histéresis. Sin embargo, no ocurren en ninguno de los sensores que se presentan en esta tesis.

## 1.6. Salida analógica o digital

Que la salida de un sensor sea analógica es sin duda un inconveniente, pues obliga como mínimo a añadir un conversor A/D, y muy probablemente sea también necesario usar alguna circuitería de acondicionamiento de señal. Y todo esto significa componentes adicionales a la FPGA. Incluso si se construye el conversor A/D con la FPGA, como se describe en [Xil99], siempre son necesarios unos pocos componentes analógicos. La única solución podría venir con el uso de FPGAs mixtas analógicas/digitales, como FIPSOC [Fau97], pero no parece que esta alternativa vaya a tener continuidad, por los costes de aunar dos procesos de fabricación muy diferentes.

Aparte de los componentes adicionales que necesitan, los sensores con salida analógica presentan otro problema, y es la relación señal/ruido. Todos los diseñadores saben que no es una tarea sencilla embeber circuitos analógicos dentro de un sistema digital, sobre todo si se quiere mantener una buena relación señal/ruido. Y esto es especialmente cierto en los sistemas actuales, con señales digitales conmutando a decenas (o incluso centenas) de MHz. Por ejemplo, hay circuitos como el MAX 1619 (descrito en el punto 2) en los que un acoplamiento de sólo 100 mVpp a 50 MHz puede provocar errores de 8 °C en la medida. Y cualquier diseñador sabe que 100 mVpp de ruido no es algo ni mucho menos extraordinario.

Aunque las técnicas para evitar estas interferencias son perfectamente conocidas (fuentes y planos de alimentación separados, filtros, desacoplos...), no deja de ser cierto que representan un inconveniente por el incremento tanto de área como de complejidad que causan en el circuito.

## 1.7. Necesidad de circuitos analógicos adicionales

Este es un punto muy relacionado con el anterior (puede considerarse equivalente); en el caso de que la respuesta del sensor sea analógica no sólo hará falta un conversor A/D, normalmente se necesitarán otros componentes adicionales:

- Polarización, p.ej. fuentes de corriente o de tensión.
- Acondicionamiento de señal, p.ej. amplificadores o eliminadores de offset

Compensación, p.ej. de la temperatura ambiente o de variaciones en Vcc.

## 1.8. Autocalentamiento

El autocalentamiento es el incremento de temperatura que provoca el sensor por la potencia que disipa durante su funcionamiento. No puede considerarse estrictamente como un error, pues no es que el sensor dé una lectura más alta que la real, sino que en efecto la temperatura se ha incrementado en el silicio como consecuencia del operación del sensor.

Es muy fácil mantenerlo en niveles razonables (por debajo de 1 °C). Sólo hay que reducir la potencia disipada por el sensor (por ejemplo, bajando su corriente de polarización o su frecuencia de operación), y lo que es más importante, activándolo sólo cuando haya que hacer una medida. Las constantes de tiempo térmicas dominantes en un circuito (centenas de milisegundos o incluso segundos) son órdenes de magnitud mayores que los tiempos necesarios típicamente para tomar una medida de temperatura (decenas o centenas de microsegundos). Por esta razón no tiene sentido tener continuamente activados los sensores, pues lo único que se conseguirá es aumentar el autocalentamiento.

## 1.9. Área

Cuando se usa un sensor integrado en el chip para medir su temperatura, una característica que no puede olvidarse es cual es el área que gasta. Y no sólo esto, sino si es necesario usar pines para obtener el resultado, o se puede obtener a través de algún otro medio (CPU integrada, puerto de configuración, JTAG...). Normalmente será este último dato el que sea más problemático, pues los pines de E/S suelen ser un recurso escaso, más que los $mm^2$ de silicio.

## 1.10. Otros parámetros

Un parámetro especialmente importante es el ancho de banda del sensor; sin embargo en esta tesis no tendrá especial relevancia porque se ha centrado en medidas estáticas. Sin embargo, para cualquier estudio dinámico (esto es, que tenga en cuenta la evolución en el tiempo) de la temperatura en un chip será un dato imprescindible.

Otro parámetro importante es el rango de operación; aunque en este caso en particular no es determinante porque está limitado por el margen de temperaturas a las que puede trabajar el circuito integrado, que es en todos los casos mucho menor que el

que ofrece el sensor. También es importante la repetitividad (que el sensor de siempre la misma salida) y la reproducibilidad (que la respuesta del sensor no cambie con el experimento). En el contexto de esta tesis, interesará que los transductores se comporten igual aunque se cambie de dispositivo o de PCB.

Por último, otros datos a tener en cuenta es si hace falta tener acceso directo al silicio para realizar la temperatura (lo cual excluye inmediatamente a los circuitos comerciales); y si el sensor es apropiado para hacer mapas térmicos.

## 2. Diodos embebidos en el circuito

Este es el método habitualmente ofrecido por los fabricantes de circuitos integrados para monitorizar la temperatura del silicio. Consiste en añadir un diodo al circuito, que se conecta al exterior usando dos pines del encapsulado. La curva I-V de un diodo viene definida por la siguiente ecuación:

$$I=I_S\left(e^{\frac{qV}{nKT}}-1\right)$$

Donde $I_S$ es la corriente de saturación, que viene determinada por los detalles constructivos del diodo (geometría, niveles de dopaje); q es la carga del electrón, $1{,}6\cdot10^{-19}$ Cb; n es el factor de idealidad (1 para un diodo ideal); K es la constante de Boltzmann, $8{,}62\cdot10^{-5}$ eV·°K$^{-1}$; y T es la temperatura en °K.

Como se puede ver, la formula tiene una dependencia exponencial inversa con T. Pero si se despeja en vez de la corriente el valor de la tensión en la unión se obtiene:

$$V=n\frac{K}{q}T\ln\left(\frac{I}{I_s}\right)$$

Que es tiene una dependencia lineal con T, mucho más sencilla de manejar. Por eso para medir la temperatura con un diodo lo que se hace habitualmente es conectarlo a una fuente de corriente constante, y observar la variación de la tensión en sus terminales, que será proporcional a T. Por ejemplo, esta es la respuesta del diodo embebido en una FPGA:

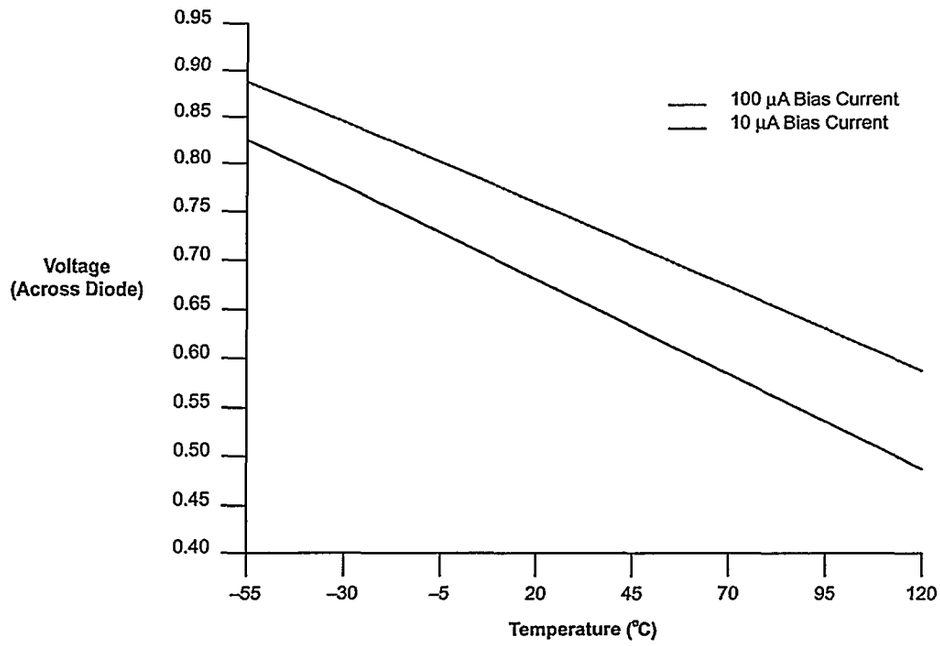



Fig. 18: Respuesta del diodo embebido en la familia Stratix de Altera [Alt02]

Pero existe todavía una alternativa mejor, y es medir la tensión para dos corrientes diferentes $I_a$ e $I_b$:

$$T = \frac{V_a - V_b}{n\frac{K}{q}\ln\left(\frac{I_q}{I_b}\right)}$$

De esta manera eliminamos la dependencia con $I_S$, de tal manera que la fórmula es válida para cualquier diodo, independientemente de cómo esté construido. Por supuesto, al hablar de diodos también se esta incluyendo las uniones base-emisor de los transistores bipolares.

Esta solución es muy común en circuitos comerciales. Se emplea en todos los microprocesadores de consumo: Intel [Int01], AMD [AMD03], e incluso en los chips gráficos de altas prestaciones: [Asu02]. Por supuesto, también se emplea en FPGAs: lo utilizan tanto Xilinx, en toda su familia Virtex (Virtex, Virtex-E, Virtex-II, Virtex-II Pro), como Altera, en su familia Stratix (Stratix, Stratix GX).

Una de las ventajas de ser un método tan usado es que existen en el mercado soluciones que implementan en un chip todos los componentes que hacen falta para medir la temperatura con esta técnica. Básicamente, estos circuitos contienen una fuente de corriente, todo el acondicionamiento de señal necesario, el conversor A/D, y una interfaz para un microprocesador. Como ejemplo se podría mencionar el MAX1619; en la siguiente figura [Max99] se muestra su conexionado típico:

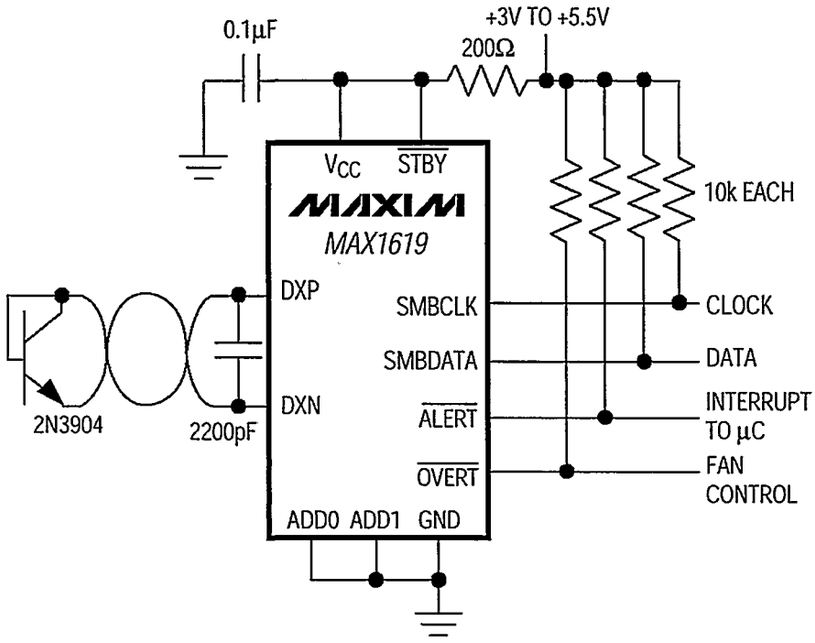



Fig. 19: Uso del MAX1619 para medir la temperatura usando un diodo embebido en el silicio

La lectura de la temperatura se realizar a través de un bus serie síncrono, necesitando sólo dos patas: CLOCK y DATA. El circuito hace todos los cálculos necesarios, devolviendo la temperatura directamente en °C. Además incluye salidas de alerta programables, una para activar un ventilador, y la otra, para en el caso de entrar en zona de peligro, generar una interrupción. Por lo demás, se puede ver que estos circuitos representan una opción muy sencilla y compacta para medir la temperatura.

Como ya se mencionó al comienzo de este capítulo, el ruido es un problema en los sensores con salida analógica. Esto se puede comprobar en el esquemático anterior; es necesario añadir un condensador en los terminales del diodo, y además la alimentación se hace a través de un filtro RC paso bajo, para eliminar el ruido de alta frecuencia de

Vcc. Pero incluso con todo esto el ruido sigue siendo un problema, como se puede ver en la siguiente figura ofrecida en la hoja de datos del fabricante, que establece el error de temperatura que puede causar el ruido:

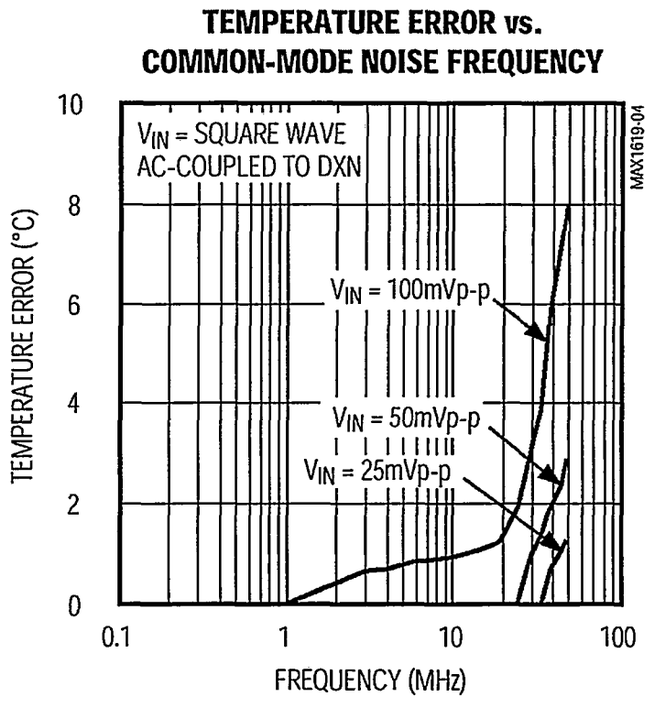



Fig. 20: Relación entre el ruido y el error de medida en el MAX 1619

Como se puede observar, niveles de ruido relativamente bajos, pero a frecuencias muy comunes en los sistemas actuales, son capaces de provocar errores considerables en la medida.

Otro de los inconvenientes de esta técnica es que no se puede utilizar para realizar mapas térmicos. Aunque nada impide crear un circuito a medida que tenga una matriz de diodos, que permitiesen obtener una termografía del chip, lo cierto es que todos los dispositivos comerciales que usan esta técnica incluyen sólo un sensor. Y suponiendo que se implementase la matriz de sensores, el gasto en pines sería enorme (dos por diodo). O habría que recurrir a un complicado multiplexado analógico.

Siguiendo con este tema, como los dispositivos comerciales sólo tienen un diodo, esta técnica sólo nos informa de la temperatura del chip en un punto. Pero no es válida para conocer la temperatura del punto más caliente (y mucho menos, su localización). Y esto es algo que explícitamente mencionan todos los fabricantes [Tra02]. Hay que tener en cuenta que un gradiente de 4 o 5 °C en un chip no es ni mucho menos extraño.

# 3. Osciladores en anillo

Un oscilador en anillo es un circuito formado por un número impar de inversores conectados en serie:

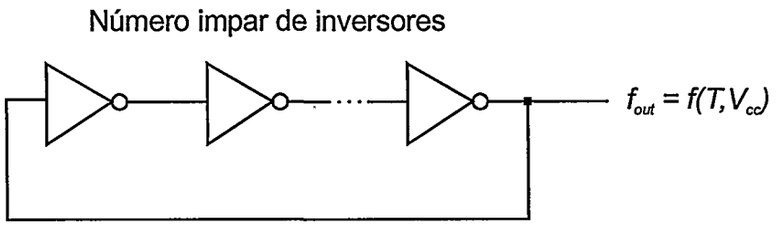



Fig. 21: Oscilador en anillo

Estos circuitos generan una onda cuadrada cuyo periodo es igual al doble del retardo de la cadena; por esta razón son muy utilizados para caracterizar los retardos de un dispositivo. De hecho, la velocidad de una determinada tecnología se suele expresar en ps/etapa de un oscilador en anillo.

En tecnología CMOS, los retardos dependen de tres factores:

- La tecnología y las variaciones en el proceso de fabricación
- La tensión de alimentación
- La temperatura del silicio

Predecir la sensibilidad del retardo respecto a estas magnitudes con los modelos de los transistores es muy complejo, más aún en las tecnologías actuales (por los efectos del canal corto). Se podrían escribir páginas y páginas de fórmulas que no aportarían demasiado, pues el modelado de los osciladores en anillo siempre se termina haciendo

de una manera empírica. Más aún, en FPGAs el problema es simplemente irresoluble, porque no se conocen los detalles físicos de implementación del oscilador (ni las características de los transistores ni incluso como están conectados), que son datos confidenciales de los fabricantes. En cualquier caso, en [Dag98] se presentan desarrollos que tratan de modelar la sensibilidad de los retardos en CMOS con respecto a la temperatura y Vcc.

En cualquier caso, en CMOS se cumple casi de manera universal que la dependencia con respecto a la temperatura es prácticamente lineal, y con una buena sensibilidad: del orden de 0,3% por °C en FPGAs [Xil95, Xil96]. Esto hace que sean excelentes candidatos a sensores de temperatura, como se verá con todo detalle en los siguientes capítulos de esta tesis. A modo de resumen, en la siguiente figura se muestra la respuesta de tres *osciladores en anillo implementados en diferentes* FPGAs:

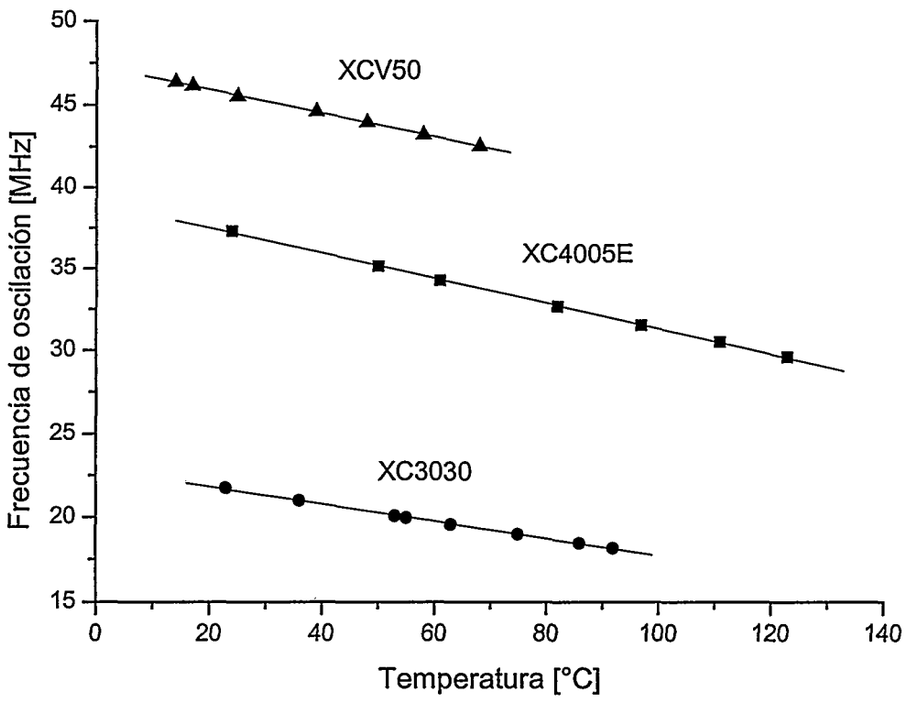



Fig. 22: Comportamiento de osciladores en anillo implementados en diversas FPGAs

El problema viene de que su dependencia con Vcc es también muy apreciable. Tanto que incluso podría ser cierta la afirmación opuesta: los osciladores en anillo son buenos sensores de la tensión de alimentación, siempre que la temperatura del chip se mantenga controlada. Afortunadamente, existen una serie de factores que mitigan este problema:

- Los circuitos de alimentación actuales, tanto lineales como conmutados, ofrecen unas precisiones muy buenas en la tensión de salida.
- En el PCB la tensión de alimentación se distribuye utilizando caminos de muy baja resistencia (normalmente planos completos), resultando en que las variaciones de Vcc a nivel de sistema son mínimas. Además, el uso de condensadores de desacoplo proporciona una ayuda adicional. Todo esto no es gratuito; viene del requerimiento de que Vcc tenga el mínimo ruido posible, algo necesario para garantizar el funcionamiento fiable del sistema. Por lo tanto se puede asegurar que todo sistema bien diseñado cumplirá estas características (si bien, según van bajando las tensiones de alimentación que emplean los circuitos, es cada vez más difícil lograrlo).
- La dependencia en la respuesta de los osciladores en anillo con respecto a las variaciones en Vcc es muy lineal, por lo que se puede compensar sin tener que recurrir a expresiones matemáticas complejas.

Pero sin duda la gran ventaja de los osciladores en anillo viene del hecho de que son circuitos completamente digitales, no necesitan de ningún componente analógico adicional. Además, los componentes necesarios para construirlos no pueden ser más sencillos, por lo que se pueden construir en cualquier dispositivo programable, tal y como se verá en el resto de esta tesis.

# 4. Termopares

Los termopares se basan en el efecto Seebeck para medir la temperatura. Descrito brevemente, si se tiene un par de conductores de distintos materiales y soldados en un extremo, en el otro extremo se crea una diferencia de potencial proporcional a la diferencia de temperatura entre el extremo soldado (habitualmente llamado punto o unión caliente) y el extremo libre (punto o unión fría).

La siguiente tabla resume las características de los principales tipos de termopares que hay estandarizados, que se denominan con una letra:

| Tipo | Material del terminal positivo | Material del terminal negativo | Coef. Seebeck @ 20°C [$\mu$V/°C] | Margen de temperaturas |
|---|---|---|---|---|
| E | Níquel 10% Cromo | Constantan (60% Cu, 40% Ni) | 62 | -100°C ~ 1000°C |
| J | Hierro | Constantan | 51 | 0°C ~ 760°C |
| K | Níquel 10% Cromo | Níquel 5% Aluminio-Silicio | 40 | 0°C ~ 1370°C |
| C | Tungsteno 5% Renio | Tungsteno 26% Renio | 24 | 0°C ~ 2320 °C |
| R | Platino 13% Rodio | Platino | 7 | 0°C ~ 1000°C |
| S | Platino 10% Rodio | Platino | 7 | 0°C ~ 1750°C |

Tabla 5: Características de los principales tipos de termopares

En la Fig. 23 se muestra la respuesta de distintos tipos de termopares. Como se puede ver, la respuesta no es demasiado lineal (salvo el caso del tipo K), aunque quizá su punto más débil sea su pobre sensibilidad. Por ejemplo, para el tipo K hacen falta 1000 °C de diferencia entre extremos para que dé una respuesta de 40 mV.

En general, este método es demasiado complicado e invasivo como para ser utilizado para medir la temperatura del silicio en circuitos comerciales. Sin embargo, se describe en esta tesis porque ha sido tradicionalmente uno de los métodos preferidos para medir la temperatura, tanto a nivel científico como industrial. Sus ventajas principales son estas:

- Amplísimo rango de temperaturas, del orden de centenas de °C, muy por encima de otros sensores
- Excelente precisión, no tanto por méritos propios (pobre linearidad y necesidad de medir la temperatura en el punto frío) sino por la tradición de la técnica, que ha propiciado que haya productos de mucha calidad en el mercado.
- No necesitan calibración previa, todas las sondas de un mismo tipo se comportan igual (porque la respuesta está solamente determinada por su composición química)

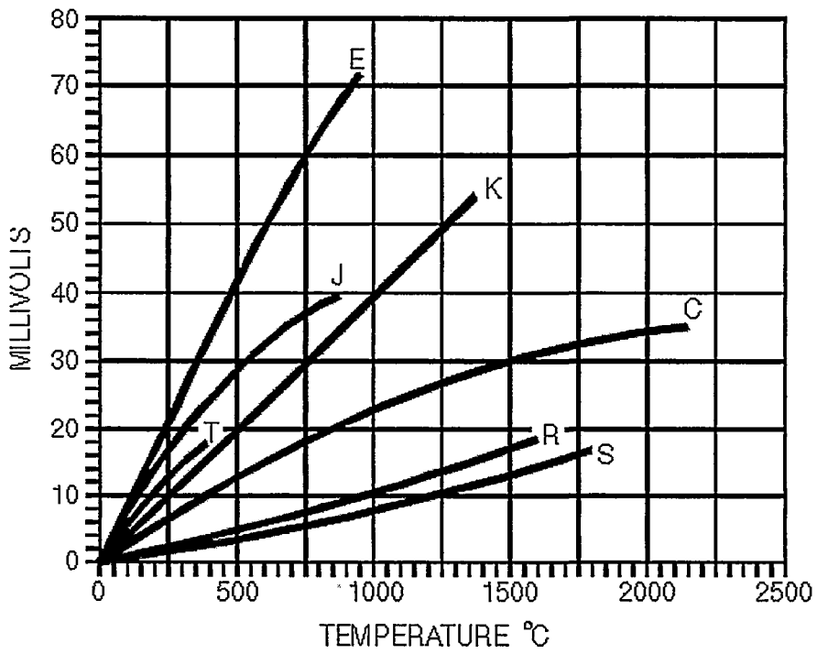



Fig. 23: Respuesta de los principales tipos de termopares

# 5. Cámaras de infrarrojos

Las cámaras de infrarrojos han sido tradicionalmente el método estándar para obtener mapas térmicos de los chips. Su funcionamiento es similar al de una cámara convencional, pero con las ópticas y fotodetectores adecuados para trabajar en esta banda del espectro. Son aparatos caros, en parte debido a su reducido mercado (aunque son muy empleadas en aplicaciones industriales), y también por sus inherentes dificultades técnicas. Por ejemplo, una de las complejidades de estos aparatos es que la tradicional "cámara oscura" se transforma en "cámara fría".

Las primeras generaciones de cámaras tenían fotodetectores que debían trabajar a temperaturas muy bajas, por lo que necesitaban un complejo sistema criogénico, o una fuente de nitrógeno líquido. En la actualidad, gracias a los sensores *microbolometer FPA (focal plane array)* este requerimiento ya no es necesario, y con un simple enfriamiento termoeléctrico queda resuelto el problema. En la siguiente tabla se resumen las características principales de un modelo comercial que podría ser usado para el análisis

de circuitos integrados. Aunque existen muchos fabricantes, no hay grandes variaciones en las características técnicas de los distintos modelos.

| **Modelo** | FLIR ThermaCAM S40 |
|---|---|
| **Lente estándar** | 24° x 18°, foco mínimo 0,3 m |
| **Lentes adicionales** | 200 µm Close-up (64mm x 48mm/150mm)<br>100 µm Close-up (34mm x 25mm/80mm)<br>46 µm Macro (15mm x 11mm/19mm) |
| **Sensor** | Focal plane array (FPA) uncooled microbolometer, 320 x 240 pixels |
| **Precisión** | ±2 °C ó ±2% |
| **Sensibilidad** | 0.08 °C a 30 °C |

Tabla 6: Características de una cámara infrarroja utilizable para el análisis de C.I.

En la siguientes figuras se pueden ver a modo de ejemplo la termografía de un PCB, donde se puede ver un circuito con encapsulado PQFP100 (Fig. 24, obtenida de *www.flir.com*), y la de un circuito integrado montado en un banco de pruebas (Fig. 25, obtenida de *www.x20.org*).

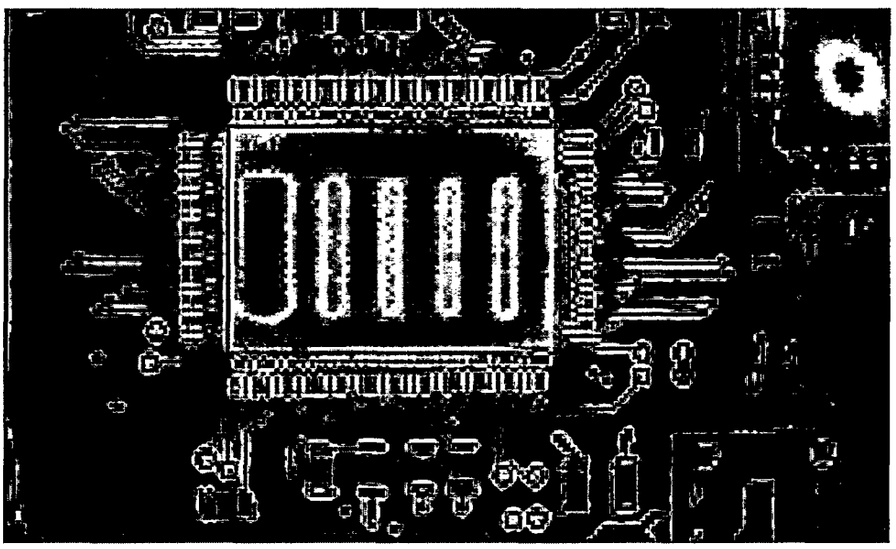


Fig. 24: Termografía de un PCB mostrando un circuito con encapsulado PQFP100

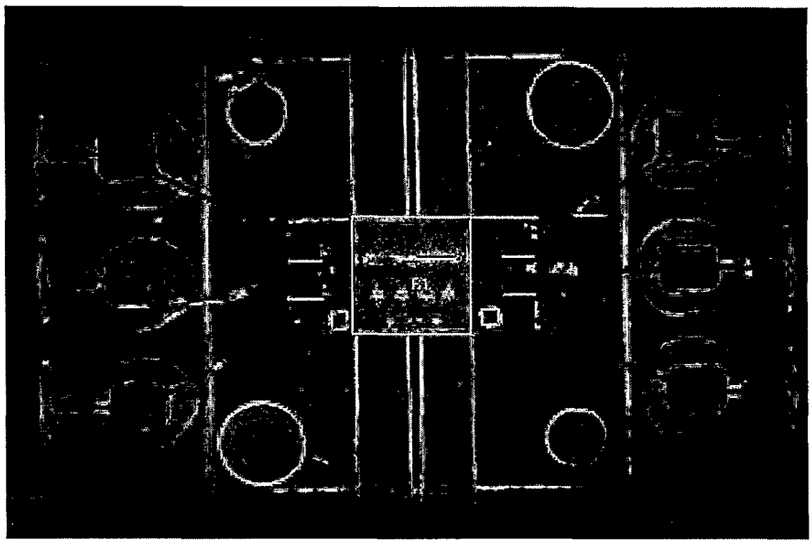


Fig. 25: Termografía de un circuito integrado

Aunque esta técnica pueda parecer muy potente, sin embargo tiene poca aplicación en el contexto de esta tesis. Y es que su mayor problema es que para poder obtener una termografía del silicio, hay que tener acceso directo a él; por lo cual es necesario eliminar el encapsulado (o usar un encapsulado que lo deje ver). Y por supuesto, no se puede emplear ningún tipo de disipador o ventilador o zócalo..., que ocultarían todavía más el chip. Como conclusión, las cámaras infrarrojas son útiles (y mucho) para el análisis a nivel de sistema (PCB), y para investigación sobre encapsulados y/o circuitos integrados (en especial con dispositivos de potencia). Pero no son válidas para analizar la temperatura del silicio en una FPGA comercial.

# 6. Otras técnicas

En los puntos anteriores se han descrito las dos técnicas más empleadas para medir la temperatura en el silicio (osciladores en anillo y diodos embebidos), y dos de las más habituales para medir la temperatura en general. Pero la cantidad de métodos diferentes de medida que hay es ingente, se podría hacer una tesis sólo con su catalogación. En este punto se hace una descripción de otras técnicas, que son comúnmente referenciadas en los artículos que tratan de la verificación térmica de los circuitos integrados.

## 6.1. Termistores

Los termistores (resistencias que varían con la temperatura) son la competencia más directa de los termopares para las medidas genéricas de temperatura. A cambio de una precisión y rango de temperaturas en general inferior que los termopares, ofrecen un manejo mucho más sencillo, pues no necesitan compensación del punto frío. Por esta razón son la técnica de medida de temperatura más común en aplicaciones industriales.

Hay muy distintos tipos. Los hay compuestos por óxidos de metales de transición (manganeso, cobalto, níquel o cobre), y dependiendo de si su resistencia aumenta o disminuye con la temperatura se llaman respectivamente PTCs o NTCs. Las hay compuestas por hilos de platino, que ofrecen una excelente precisión y rango de temperatura. Por último, un área de silicio dopado funciona como un termistor (su resistencia varía apreciablemente con la temperatura), por lo que se podría usar como sensor integrado. Pero no se emplea en la práctica porque no es más fácil de implementar que un diodo, que además tiene mejores propiedades como sensor.

## 6.2. Otros sensores integrados

Los diodos y los osciladores en anillo no son los únicos transductores de temperatura que se pueden integrar en un circuito integrado. Hay otros sensores que pueden construirse en tecnología CMOS, como el conversor temperatura-corriente descrito en [Sze97] y [Sya01]. En este caso, la salida analógica puede convertirse a digital usando un oscilador en anillo controlado por corriente. Otra alternativa interesante son los sensores diferenciales; que no informan de la temperatura absoluta en un punto, sino de los gradientes de temperatura. Como se verá en el capítulo 6, incrementos puntuales del consumo (por ejemplo, un cortocircuito) causan aumentos locales de la temperatura en el circuito integrado, que podrían ser detectados con esta técnica. Este método está descrito en [Alt02]; en este artículo también se detallan muy bien las técnicas de los dos siguiente puntos.

## 6.3. Microscopio de fuerzas atómicas

Esta técnica se basa en sustituir la punta de un microscopio convencional de fuerza atómica (AFM) por un termistor de platino. El funcionamiento de un AFM consiste en hacer un barrido de la muestra con una punta muy afilada (prácticamente acaba en un

único átomo). Cuanto más cerca esté la punta de la muestra, mayor será la fuerza atómica de repulsión a la que se ve sometida. La punta está montada en un cabestrante que permite conocer, a través de la deflexión de un haz láser, la magnitud de esta fuerza. Para obtener la imagen se hace un escaneado de dos dimensiones, durante el cual se va subiendo o bajando la muestra para mantener constante la fuerza atómica (o sea, la distancia de la punta). De esta manera se llega a conocer la altura de la muestra con un detalle inmenso, pudiendo incluso llegar a observarse como se disponen los átomos en su superficie.

Esta técnica no sólo permite obtener la información topográfica de la muestra, como cualquier otro AFM, sino que midiendo la resistencia del hilo de platino que forma la punta se puede también conocer la temperatura de la superficie. El precio a pagar es que la resolución la medida topográfica no es tan buena como en los AFMs usados para visualizar estructuras atómicas, pues el hilo de platino no resulta una punta muy afilada; aunque en cualquier caso es muy buena, de 50 nm, mucho mejor que en la de un microscopio óptico.

## 6.4. Técnicas basadas en láser

Básicamente hay dos: las en el refractómetro y en el interferómetro láser. La primera mide la temperatura directamente, basándose en que el coeficiente de refracción de un sólido depende de su temperatura: se lanza un haz láser a la muestra, y se mide la luz reflejada. La segunda es una técnica desarrollada para obtener una información topográfica de la muestra, lo mismo que el AFM. En este caso la temperatura se mide indirectamente, a través de las deformaciones en la muestra que causa la dilatación por calentamiento. Aunque las variaciones son muy pequeñas, menos incluso de un nanómetro, la gran resolución de este método (un femtometro) permite observarlas sin problema.

La resolución espacial de estos dos métodos es similar, 1 $\mu$m, y su gran ventaja es su gran ancho de banda, 150 MHz (aunque este dato no sea relevante en esta tesis, que se centra en las medidas estáticas). Al igual que la técnica basada en el AFM, la gran limitación de estos dos métodos es que hay que tener acceso directo al silicio, lo cual limita su utilización a trabajos de investigación o a la verificación antes del encapsular el chip (aunque esta posibilidad podría ser discutible si se usa montaje *flip-chip*).

# 7. Conclusiones

En este capítulo se ha presentado las principales propiedades que sirven para caracterizar un transductor de temperatura, y se han descrito los cuatro sensores más comunes, más otros cinco métodos también referenciados en la literatura, pero menos comunes:

- El diodo integrado es la técnica estándar empleada por los fabricantes de los circuitos integrados. Sus ventajas son su sencillez de manejo y que ofrece una buena precisión. Sus mayores problemas vienen de que su salida es analógica, por lo que necesitan de un circuito externo, y que deben ser integrados por el fabricante; no se pueden crear con los recursos estándares de la FPGA. Por esta razón (el fabricante sólo incluye uno) no se pueden emplear para realizar mapas térmicos.
- Los osciladores en anillo son la solución a estos inconvenientes: su salida es digital, y como se pueden crear con los recursos estándares de la FPGA, se pueden añadir tantos como quepan en ella. Su inconveniente es que son muy sensibles a las variaciones de la tensión de alimentación.
- Los termopares son el método más común para medir la temperatura en experimentos físicos, y se utilizan mucho en la investigación de los aspectos térmicos de los circuitos integrados: para medir la temperatura ambiente, o del encapsulado... Pero no son válidos para medir la temperatura del silicio. La alternativa a los termopares como sensores generalistas son los termistores, que tampoco se suelen emplear integrados en un chip (aunque en este caso si sería posible hacerlo).
- Los manera más usual (e inmediata) de obtener un mapa térmico es utilizar una cámara de infrarrojos. Pero este método no es útil en circuitos comerciales como las FPGAs, porque exige tener acceso visual directo al silicio. Esta limitación no es exclusiva de este método, otros como el AFM o las técnicas basadas en láser también la sufren. Esto deja a los osciladores en anillo como única posibilidad de hacer mapeado térmico en FPGAs.

# Capítulo 4.

# Una metodología de verificación térmica en FPGAs

En este capítulo se exponen las ideas centrales desarrolladas en esta tesis para medir la temperatura en circuitos reconfigurables. Se muestran las principales características de cada una de las alternativas analizadas, se describe la metodología de calibración y se resumen los compromisos de diseño de cada solución.

En la sección 1.1 se muestran las principales características de los osciladores en anillo utilizados como circuitos de referencia para el resto de los experimentos. En la siguiente sección se analiza la opción de los diodos de enclavamiento. Posteriormente (sección 3) se desarrolla una metodología para la calibración de los dispositivos.

## 1. Osciladores en anillo como sensores de temperatura

La idea central de esta tesis puede resumirse en los siguientes hechos:

1. La temperatura es una magnitud analógica.
2. Las FPGAs, salvo breves excepciones, son chips digitales.
3. La única magnitud analógica que puede medirse mediante un circuito digital es la frecuencia.
4. Y la frecuencia de salida de los osciladores en anillo depende, en otras magnitudes, de la temperatura.

Todo esto lleva a la conclusión de que los osciladores en anillo pueden ser unos sensores de temperatura ideales para ser empleados en los circuitos reconfigurables, pues se pueden construir sólo con los recursos de la FPGA, sin tener que recurrir a circuitos analógicos externos,

A modo de recordatorio del capítulo anterior, estos circuitos están compuestos por un lazo cerrado que incluye un número impar de inversores. De este modo, se produce el cambio de fase necesario para mantener la oscilación (Fig. 26). El periodo resultante es dos veces la suma de los retardos de todos los elementos que componen el lazo. Los inversores pueden mapearse utilizando las tablas de *look-up* de los bloques lógicos configurables (CLBs), o los propios inversores que incluyen los bloques de E/S (IOBs) de las FPGA. En cualquiera de los casos, es útil insertar una puerta AND para abrir el lazo y evitar el autocalentamiento, así como un buffer a la salida, con el objeto de prevenir variaciones de frecuencia debido a variaciones en la carga de la señal $f_{out}$.

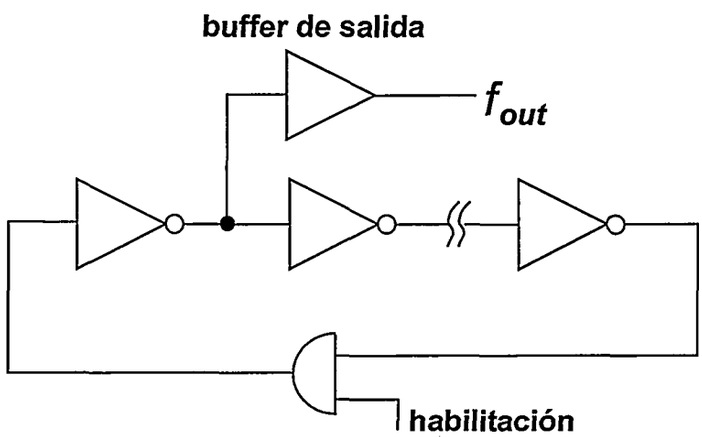



Fig. 26: Esquema general de un oscilador en anillo (*ring-oscillator*).

Como ya se esbozó en el capítulo anterior, las ventajas de los osciladores como transductores térmicos en circuitos programables son múltiples:

1. Requieren pocos elementos de una FPGA.
2. Algunas familias de FPGA ya incluyen osciladores internos.

3. Al igual que otros sensores internos, miden la temperatura de la unión en lugar de la del encapsulado.
4. Todas las señales del circuito son digitales, no necesitándose ningún componente externo analógico, ni conversores A/D.
5. Al ser la salida digital, está libre de los problemas de ruido que ocurren con las señales analógicas.
6. Los sensores ocupan poca área. Uno o dos CLBs o incluso un único IOB.
7. Un sensor o incluso un *array* de ellos puede ubicarse en cualquier posición de la pastilla, haciendo posible realizar la construcción de un mapa térmico del dado.
8. Sólo se requieren dos patas de la FPGA.
9. Los sensores pueden insertarse, moverse, o eliminarse dinámicamente.

En una FPGA, un oscilador en anillo puede construirse manualmente o usando las herramientas de PPR automático que incluyen todos los fabricantes. En este último caso, se deben incluir directivas de alto nivel para evitar la simplificación de parejas de inversores durante el proceso de simplificación lógica. En cualquier caso, es útil situar los inversores sobre LUTs distantes con el objeto de aumentar los retardos de interconexión. El objetivo en este caso, inusual en diseño digital, es disminuir la frecuencia del funcionamiento para minimizar el problema de autocalentamiento y el tamaño del contador de frecuencia.

## 1.1. Circuitos de prueba

Durante la realización de esta tesis se construyeron y caracterizaron un total de 15 osciladores en FPGAS de Xilinx, de las series XC3000 y XC4000. Las tablas adjuntas resumen las principales características de los mismos. Los valores de retardos han sido extraídos de la información proporcionada por el analizador estático de tiempos. Los circuitos comprenden versiones con diferentes relaciones entre los retardos combinacional y de rutado, ubicación sobre el dado y tipo de FPGA, con el objeto de determinar si alguno de esos factores producen algún efecto especial en la respuesta con la temperatura. Debido a la sencillez de los circuitos, todos ellos fueron creados manualmente sobre el editor de FPGAs.

### *1.1.1. Circuitos de prueba sobre XC4005E*

| Circuito | Característica | Número de inversores | Retardo del bloque combinacional [ns] | Retardo del rutado [ns] |
|---|---|---|---|---|
| Ring1 | Retardo de rutado dominante. 2 CLBs, 4 LUTs tipo F/G más 2 tipo H | 3 | 12,6 | 26,0 |
| Ring2 | Retardo combinacional y de rutado similar. 2 CLBs, 4 LUTs tipo F/G más 2 tipo H | 3 | 12,6 | 11,6 |
| Ring3 | Retardo combinacional dominante. 2x2 CLBs, 8 LUTs tipo F/G más 4 tipo H | 7 | 25,2 | 11,9 |
| OSC4 | Oscilador interno disponible en la serie XC4000 | - | - | - |
| Ring5 a Ring8 | Una matriz de 4 sensores idénticos, situados en los ángulos del dado. Cada sensor tiene alto retardo combinacional y ocupa 2x2 CLBs. | 3 | 23,2 | 11,8 |

Tabla 7: Principales características de los osciladores [Lop98, Lop00]

Todos estos circuitos fueron implementados en una FPGA del tipo XC4005EPC84-3C. En las siguientes figuras se puede muestra el layout de Ring1 (Fig. 27), Ring2 (Fig. 29), Ring3 (Fig. 30) y OSC4 (Fig. 28). Por otro lado, en la sección 1.2 se detalla la matriz de osciladores Ring5 a Ring8.

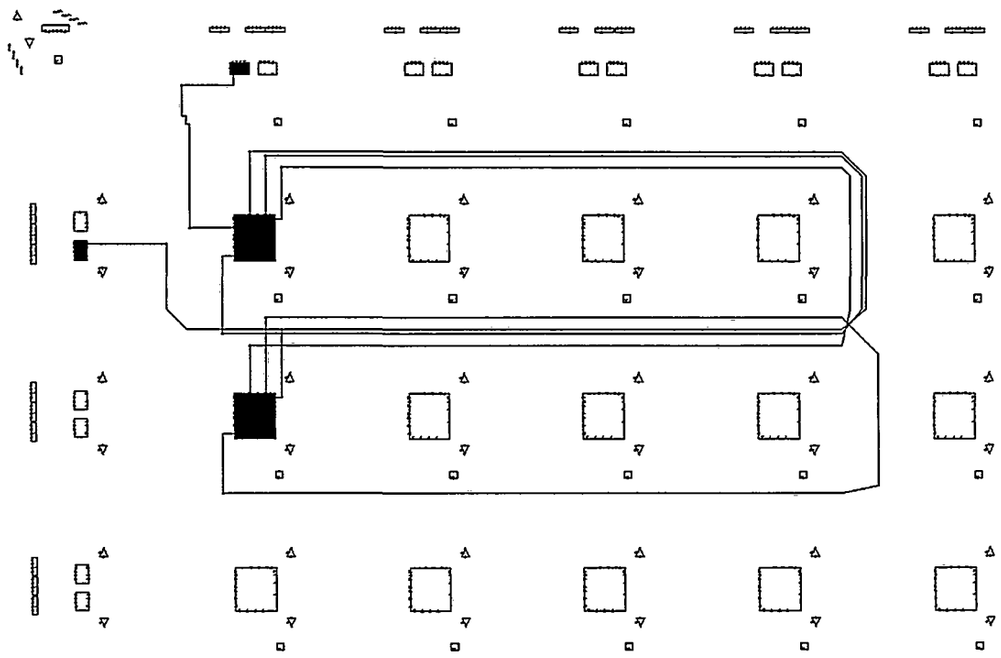


Fig. 27: Layout del circuito de pruebas Ring1

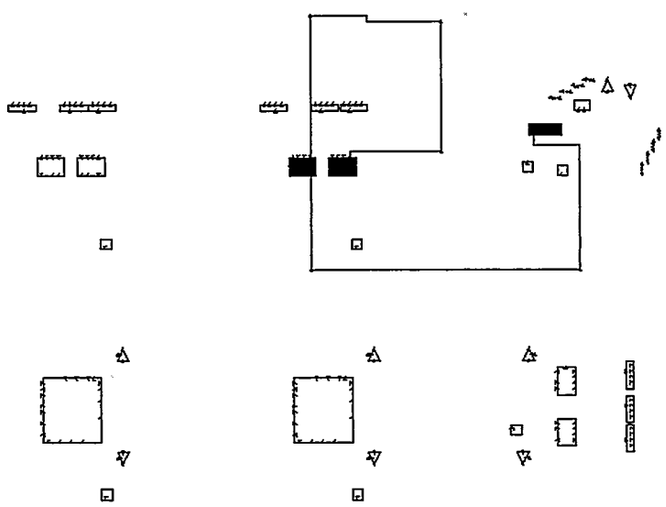


Fig. 28: Layout del circuito de pruebas OSC4

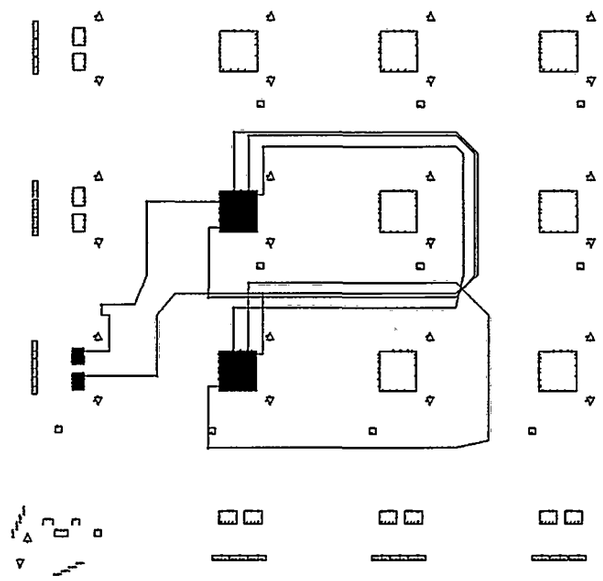


Fig. 29: Layout del circuito de pruebas Ring2

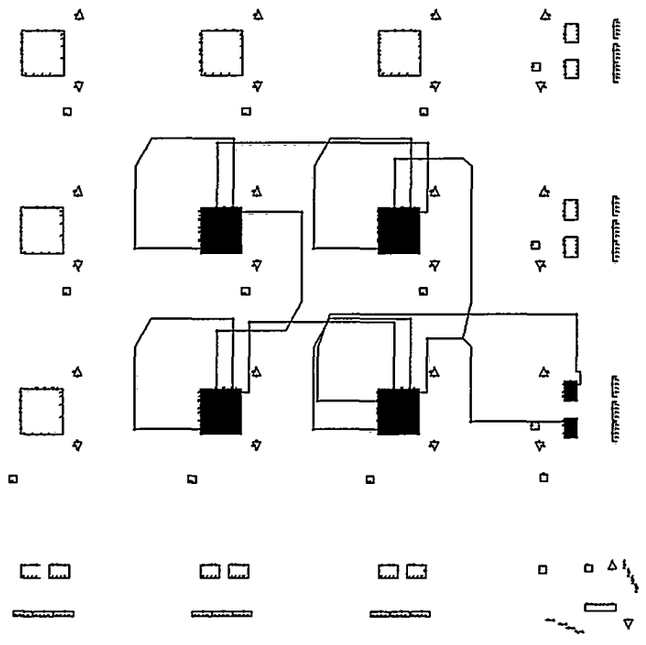


Fig. 30: Layout del circuito de pruebas Ring3

### *1.1.2. Circuitos de prueba sobre XC4005*

| Circuito | Característica | Número de inversores | Retardo del bloque combinacional [ns] | Retardo del rutado [ns] |
|---|---|---|---|---|
| Ring9 | Retardo de rutado dominante. 2 CLBs, 4 LUTs tipo F/G más 2 tipo H | 3 | 24,0 | 47,8 |
| Ring10 | Retardo combinacional y de rutado similar. 2 CLBs, 4 LUTs tipo F/G más 2 tipo H | 3 | 24,0 | 20,1 |
| Ring11 | Oscilador basado en un IOB. El lazo se abre utilizando el control triestado del IOB | 1 | 9,0 | 10,8 |
| OSC4b | Oscilador interno disponible en la serie XC4000. | - | - | - |

Tabla 8: Principales características de los osciladores [Boe97, Lop97].

Estos sensores se implementaron en una FPGA más lenta y antigua, una XC4005PC84-6C. Los osciladores Ring9 y Ring10 tienen un layout idéntico a los ya mencionados Ring1 y Ring2; el objetivo ha sido observar si existen diferencias producidas por el cambio de tecnología. Ring11 está implementado con sólo un IOB; utiliza como único inversor la opción de "activo a nivel bajo" del buffer de salida de ese bloque. Su retardo combinacional corresponde a la suma de retardos de las primitivas OBUF e IBUF. En la Fig. 31 se representa su layout, y en la Fig. 32 se puede observar como ha sido programado el IOB. Aunque esta solución es muy austera en términos de área, el problema es que su retardo global es considerablemente menor. Por lo tanto, su frecuencia de oscilación es mayor que para los circuitos construidos con CLBs, algo que ya se ha visto que no es recomendable. Por último, también se ha medido el comportamiento de la oscilador interno para esta FPGA; el layout en este caso es completamente idéntico al de OSC4.

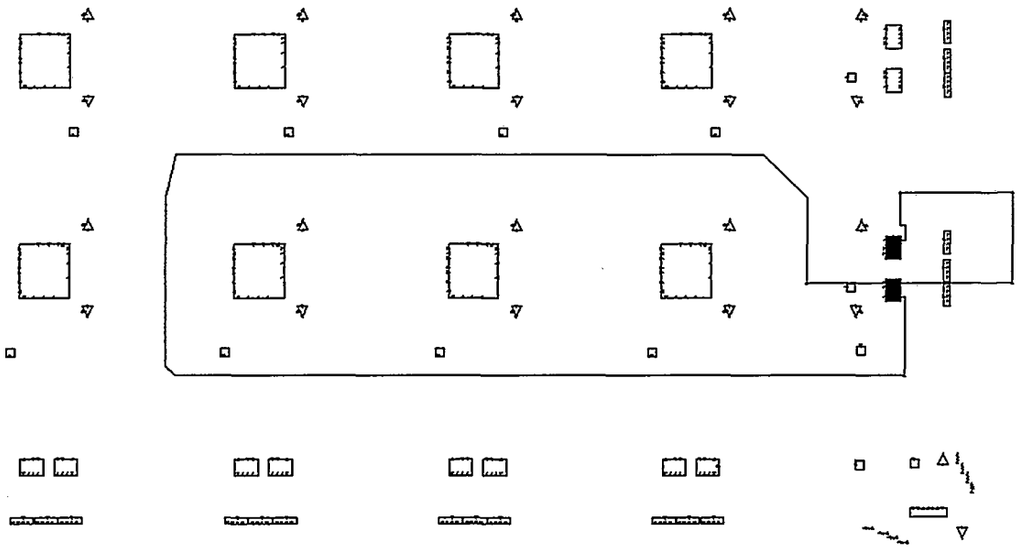


Fig. 31: Layout del circuito de pruebas Ring11

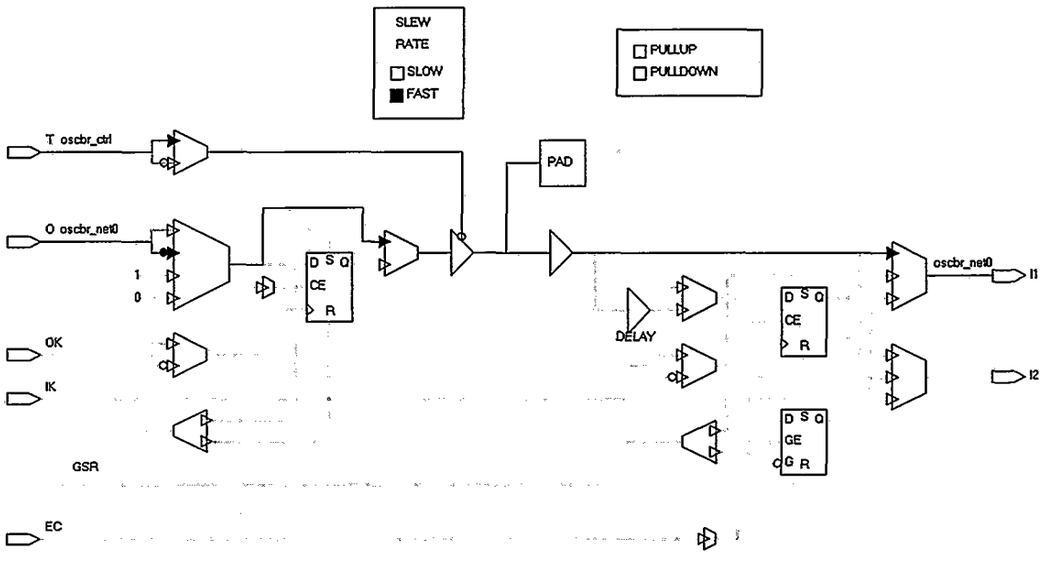



Fig. 32: Detalle de la configuración del IOB en Ring11

### *1.1.3. Circuitos de prueba sobre XC3030*

| Circuito | Característica | Número de inversores | Retardo del bloque combinacional [ns] | Retardo del rutado [ns] |
|---|---|---|---|---|
| Ring12 | Retardo de rutado dominante, 2 CLBs, 4 LUTs | 3 | 22,0 | 50,7 |
| Ring13 | Retardo combinacional y de rutado similar, 2 CLBs, 4 LUTs | 3 | 22,0 | 14,4 |
| Ring14 | Retardo combinacional y de rutado similar, 2 CLBs, 4 LUTs. El rutado emplea 3 *long-lines* y 1 *direct-line* | 3 | 22,0 | 17,9 |
| Ring15 | Oscilador basado en un IOB. El lazo se abre utilizando el control triestado del IOB | 1 | 8,0 | 12,0 |

Tabla 9: Principales características de los osciladores [Boe97].

Estos circuitos se corresponden con los primeros experimentos, que se realizaron en una XC3030PC84-125 (ver nota[1]). Ring 12 (Fig. 33) y Ring13 (Fig. 34) son las soluciones más generales, construidas a base de CLBs y con diferentes longitudes de rutado. Ring14 (Fig. 35) es muy parecido a Ring13, sólo que emplea *long-lines*. En la familia XC3000 los recursos de rutado eran muy escasos, y estas líneas facilitaban la implementación de señales con alto fanout; en la figura se han resaltado estas *long-lines*. Por último, el sensor Ring15 (Fig. 36) está construido con un único IOB, y al igual que Ring11, utiliza el inversor programable que hay en el buffer de salida.

[1] Las tesis se alargan en el tiempo y ésta no ha sido la excepción. La denominación de este *chip* corresponde a la antigua normalización de Xilinx, donde "–125" indicaba la frecuencia de *toggle* del CLB. Es decir, 125 MHz. Correspondería aproximadamente a una FPGA algo más rápida que la tipo "–6" para la familia XC4000.

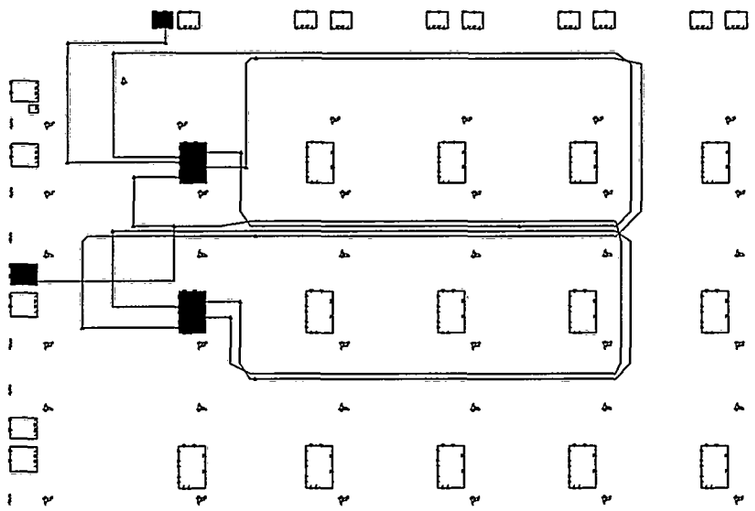


Fig. 33: Layout del circuito de pruebas Ring12

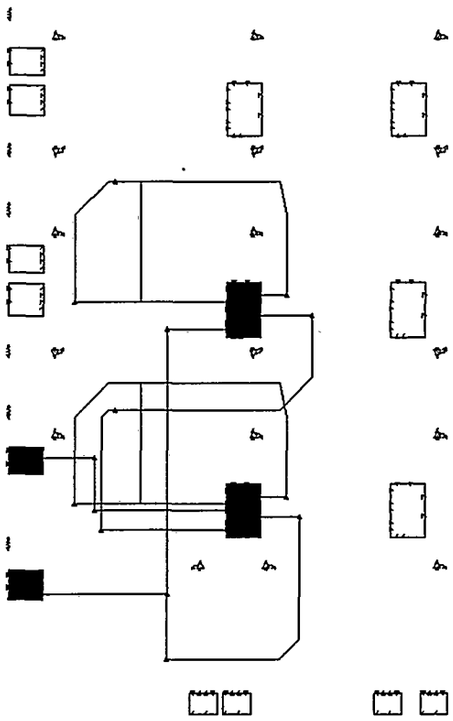


Fig. 34: Layout del circuito de pruebas Ring13

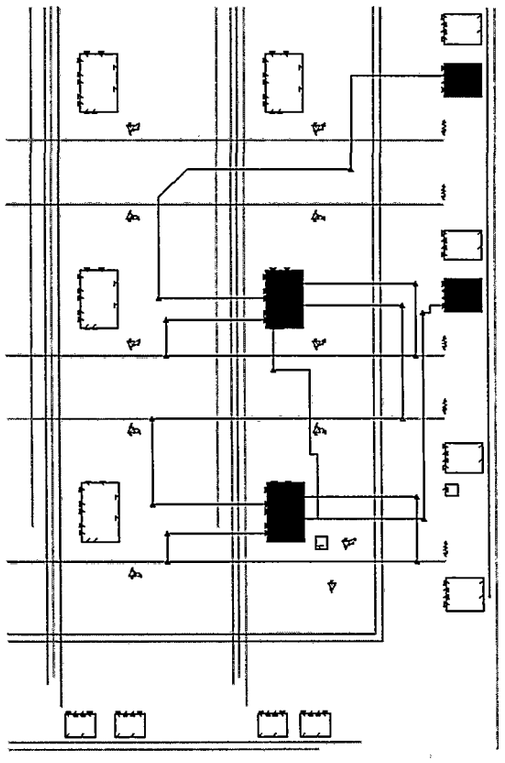


Fig. 35: Layout del circuito de pruebas Ring14

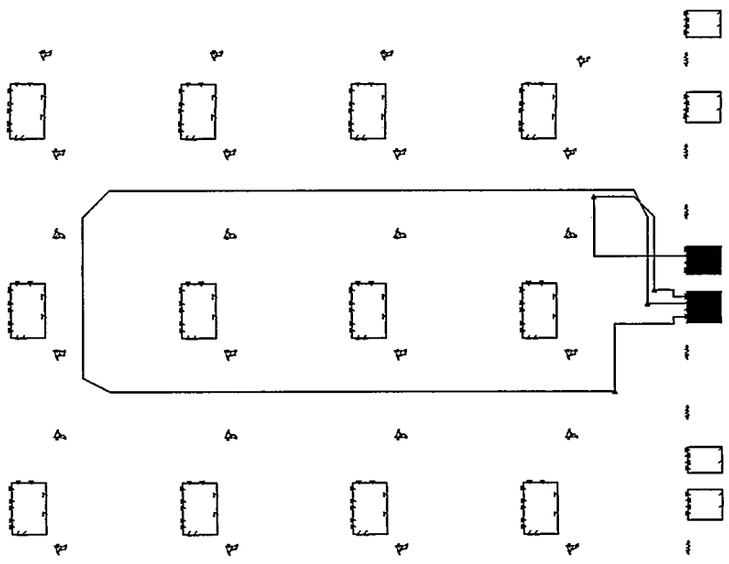


Fig. 36: Layout del circuito de pruebas Ring15

## 1.2. Matriz de osciladores

Con el objeto de estudiar el acoplamiento entre las respuestas de sensores idénticos, se construyó un conjunto de cuatro osciladores idénticos, situados en los ángulos del chip. En la Fig. 37 se muestra un esquema de la matriz, que también incluye en la esquina inferior izquierda una circuitería para llevar a cabo los experimentos sobre colisiones que se explican en la sección 5.

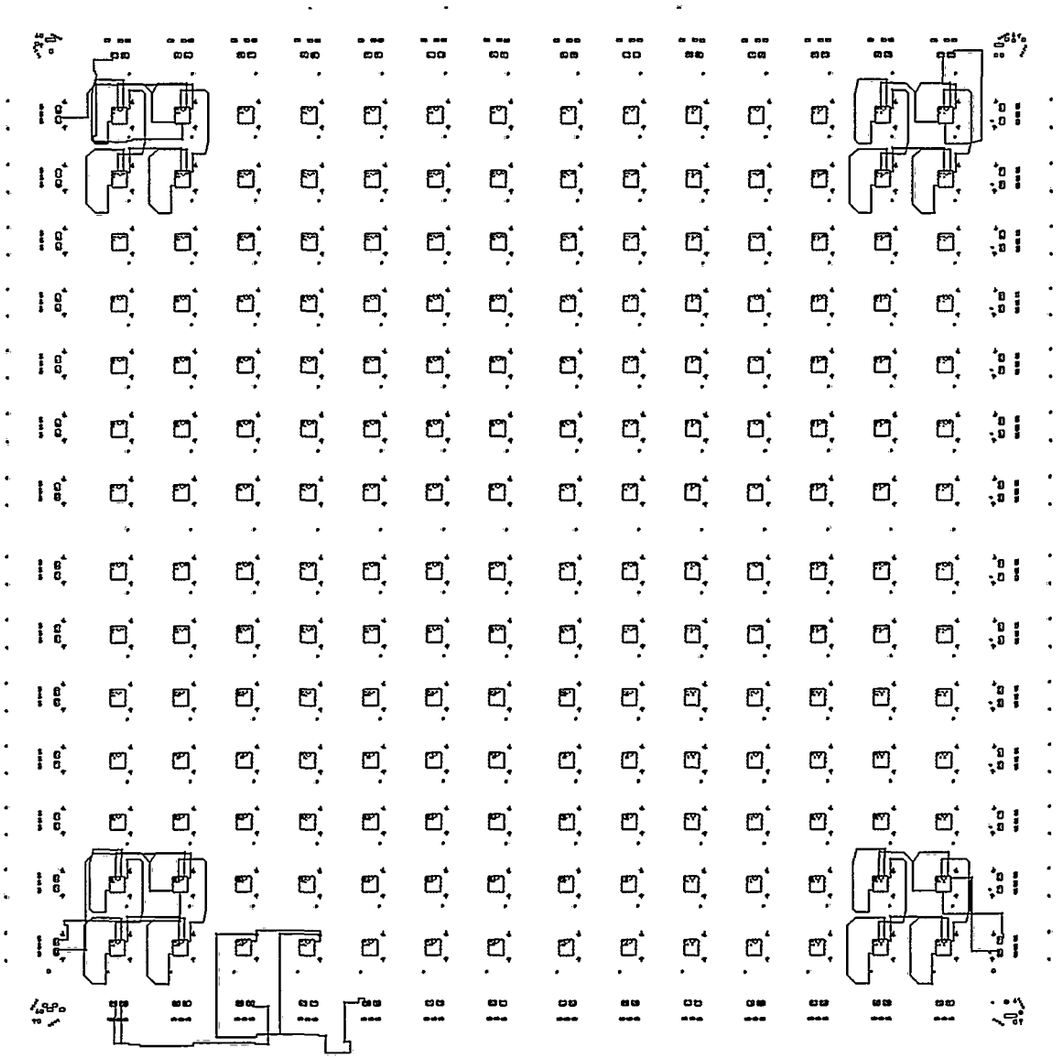


Fig. 37: Matriz de osciladores idénticos

Los osciladores fueron mapeados y rutados a mano, y utilizan 4 CLBs para aumentar su periodo de la oscilación. El retardo global del lazo es de 35 ns, de los cuales 23,2 ns que corresponden a las LUTs y 11.8 ns a la interconexión. Cada una de las salidas se pasa a través de otro CLB, que hace de buffer para prevenir posibles desbalances entre los cuatro caminos hasta las patas del chip. En la Fig. 38 se muestra el esquema detallado de uno de los circuitos.

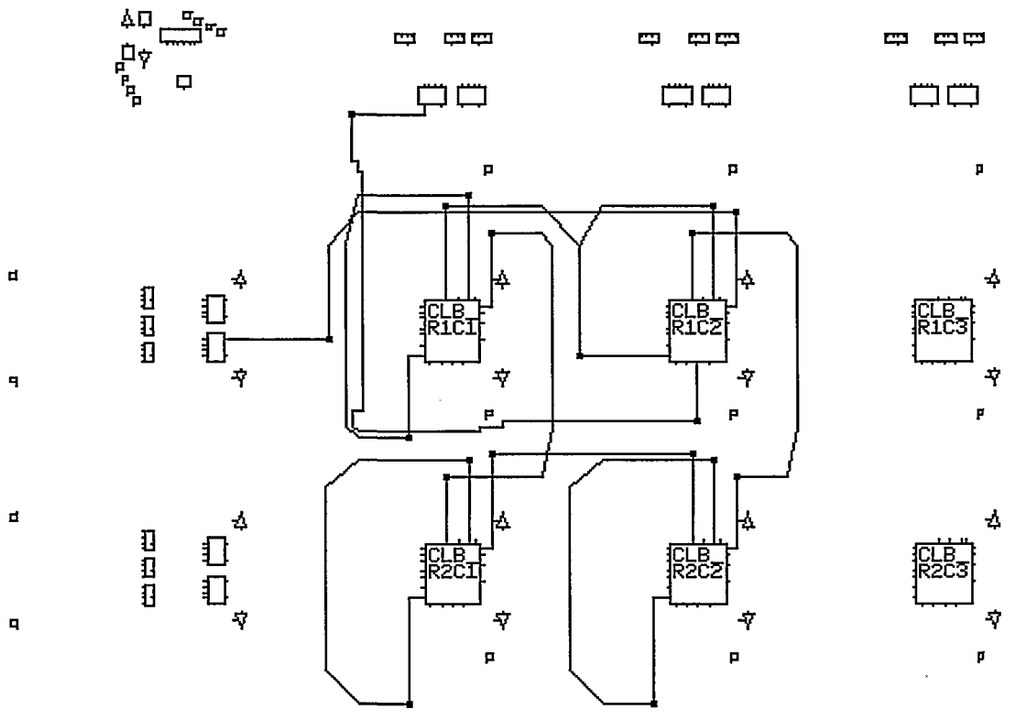
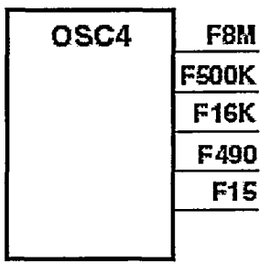



Fig. 38: Layout del circuito de prueba Ring5 de la matriz de sensores

## 1.3. Utilización de la célula OSC4 como sensor de temperatura

La célula interna OSC4, presente en la FPGAs serie XC4000 y Spartan, es un oscilador de 5 valores de frecuencia. En la figura adjunta se resumen las características del mismo [Xil99b]:




OSC4 provides internal clock signals in applications where timing is not critical. The available frequencies are determined by FPGA device components, which are process dependent. Therefore, the available frequencies vary from device to device. Nominal frequencies are 8 MHz, 500 kHz, 16 kHz, 490 Hz, and 15 Hz. Although there are five outputs, only three can be used at a time, with 8 MHz on one output and one frequency each on any two of the remaining four outputs. An error occurs if more than three outputs are used simultaneously

Fig. 39: Características principales de la célula OSC4. Extraído de [Xil99b]

En la sección 4 se demuestra que este circuito tiene un comportamiento excelente como sensor de temperatura. Como curiosidad, esta aplicación fue comunicada a Xilinx por el director de esta tesis durante una visita a la compañía en 1998. Aunque ya se estaban realizando experimentos en esta línea [Alf98], no se había descubierto esta utilidad escondida en el OSC4. Incluso el fabricante advertía a los usuarios que la frecuencia no era muy estable, justamente por el efecto de la temperatura. Por ejemplo, en el *Answer Record No. 499* de la *Answers Database* [Xil02c], se advertía: *"Given that the nominal frequency is 8 MHz, with temperature and VCC variation, the frequency would vary between 10 MHz and 4 MHz".*

En efecto, de acuerdo con las medidas realizadas en esta tesis, la célula OSC4 de una XC4005EPC84-3C oscila a 7.2 MHz a temperatura ambiente (24 °C) y a 100 °C su frecuencia ya cae a 5.8 MHz. Sin embargo, el valor mínimo de 4 MHz parece algo exagerado: se alcanzaría a una temperatura cercana a 220 °C.

# 2. La opción de los diodos de enclavamiento

Durante una discusión en un grupo de usuarios en Internet, Peter Alfke, el conocido director de aplicaciones de Xilinx, propuso otra ingeniosa alternativa para determinar la temperatura de una pastilla: usar los diodos de enclavamiento que hay en las patas de las FPGAs como sensores de temperatura [Alf97]. Con este truco se puede extender la técnica del diodo embebido a cualquier FPGA, porque sólo las familias más potentes disponen de un diodo dedicado. Todos los dispositivos programables comerciales (más aún: prácticamente todos los circuitos digitales) disponen de diodos de enclavamiento como medida para protegerse frente a descargas de electricidad estática y otras sobrecargas.

Para usar esta técnica sólo hace falta gastar una pata de la FPGA, a la que se le conecta una fuente de corriente. Por supuesto, esta fuente debe estar conectada a una tensión mayor que Vcc o menor de 0 V, porque si no es así, no entran en funcionamiento los diodos de enclavamiento. En la Fig. 40 se muestra una estructura simplificada de un IOB junto con el esquema de medida utilizado en este trabajo: una fuente de corriente que inyecta cerca de 1 mA en el diodo inferior. Obviamente, el búfer triestado debe quedar en alta impedancia, para evitar que interfiera en la medida.

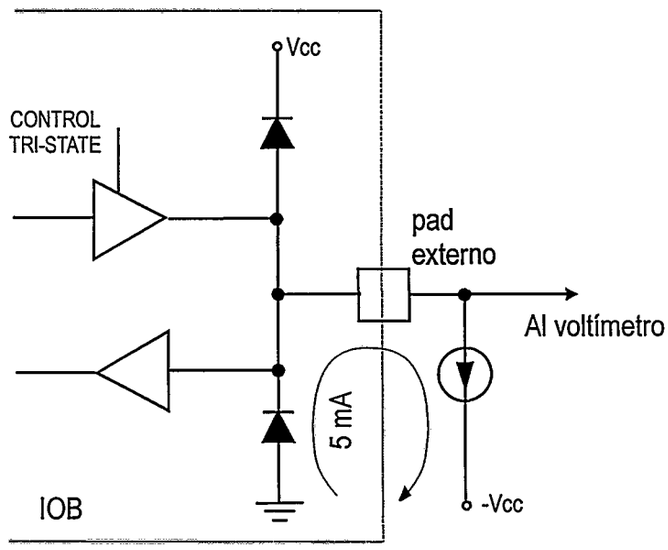



Fig. 40: Diodos de enclavamiento como sensores de temperatura. Esquema de polarización

Un refinamiento de este experimento debería probar un abanico de corrientes, con el objeto poder determinar la temperatura aunque no se conozca la corriente de saturación $I_s$ del diodo, tal y como se comentó en el capítulo anterior.

En la siguiente figura se muestra la fuente de corriente empleada en los experimentos. Como se puede ver es un circuito muy sencillo, que se basa en una referencia de precisión y una resistencia del 0,1% para proporcionar una corriente muy estable de 0,991 mA.

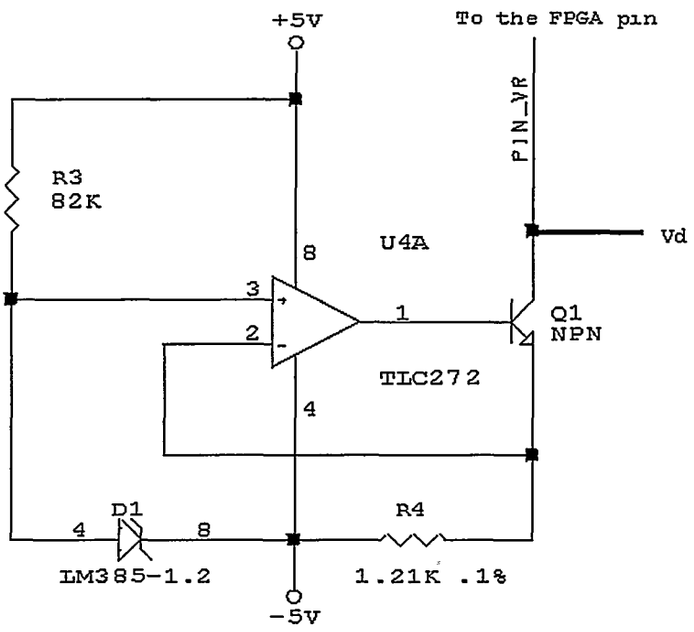



Fig. 41: Fuente de corriente. Circuito utilizado

# 3. Estrategia de calibración

Para realizar una calibración exacta, las diferentes FPGAs fueron introducidas en un horno de temperatura controlada (Fig. 42). Para ahorrar tiempo (el horno requiere largos periodos para alcanzar el equilibrio térmico), cada FPGA se configuró con cuatro sensores, aunque únicamente se habilitó uno por vez para su caracterización, una vez alcanzado el equilibrio térmico.

La temperatura del chip se determinó fijando al centro de encapsulado un termopar tipo J (Hierro-Constantan, Fe-CuNi). Para mejorar la conductividad térmica de esta unión se utilizo una resina epoxy conductora, a base de plata. Las salidas se sacaron fuera del horno mediante un cable de cerca de un metro. Para evitar que la capacidad del cable incremente el consumo de la FPGA durante las mediciones, y por tanto, el autocalentamiento, se utilizó un driver adicional tipo 74HC125.

Todo esto queda mejor resumido en la siguiente figura, donde se esquematiza el montaje utilizado para calibrar los sensores:

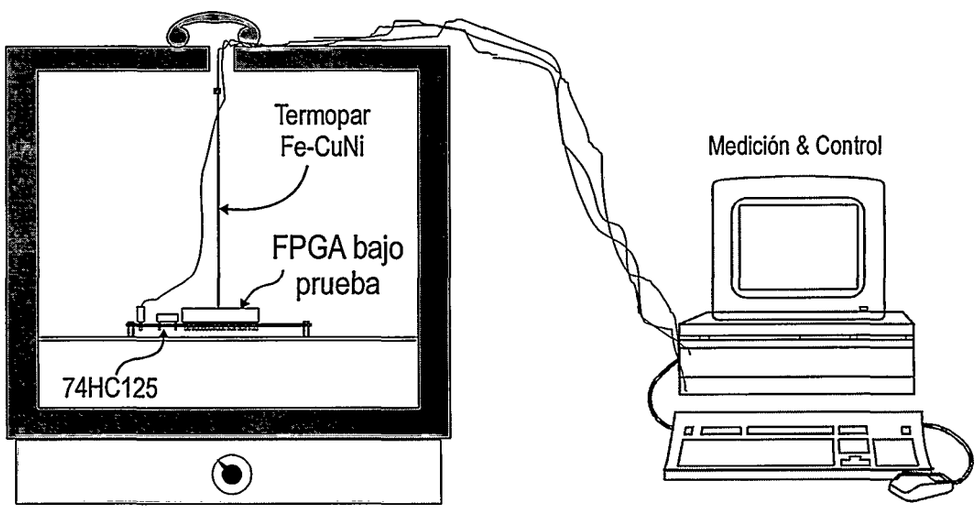



Fig. 42: Esquema de calibración de los sensores

En cualquier transductor térmico activo, el autocalentamiento producido por la potencia que disipa el propio sensor es una fuente de error que debe minimizarse. En este caso, la solución adoptada fue habitar el oscilador bajo prueba sólo durante una pequeña ventana de medición. El procedimiento (Fig. 43) se puede resumir en los siguientes pasos:

- El oscilador se habilita y se deja evolucionar durante 0.2 ms, un tiempo suficiente para que la frecuencia de oscilación supere el transitorio de arranque. En realidad, se ha medido que la oscilación se estabiliza mucho más rápido, en uno o dos periodos.
- Durante los siguientes 4 ms se mide la frecuencia de salida y posteriormente se detiene el circuito.
- El procedimiento se repite cada 250 ms. Es decir, se realizan cuatro mediciones por segundo. Esta frecuencia puede reducirse aún más considerando la velocidad de los procesos térmicos en el horno.

Todos los parámetros temporales anteriores fueron determinados empíricamente mediante la observación de varios sensores. Por supuesto, deben revisarse si se repiten los experimentos en otras FPGAs, aunque en cualquier caso pueden considerarse como un buen punto de partida.

En la Fig. 44 se demuestra la necesidad de esta estrategia. El gráfico sobrepone la frecuencia de salida de Ring5 para dos los modos del funcionamiento diferentes: habilitación en una ventana y funcionamiento permanente. Aunque la temperatura se mantuvo constante a 22°C durante la verificación de la técnica, un corrimiento de frecuencia de más de 129 KHz (equivalente a 2.5 °C) se pudo observar en el segundo modo de funcionamiento. Por el contrario, el efecto es despreciable en el modo ventana.

Para realizar todas las mediciones anteriores se desarrollo una instrumentación a medida, usando una FPGA de Xilinx tipo XC3130 para el control del oscilador y la medición de la frecuencia, y un microcontrolador 68HC11 para enviar todos los resultados a un PC. Más detalles acerca de este equipo de medida pueden encontrarse en el apéndice B.

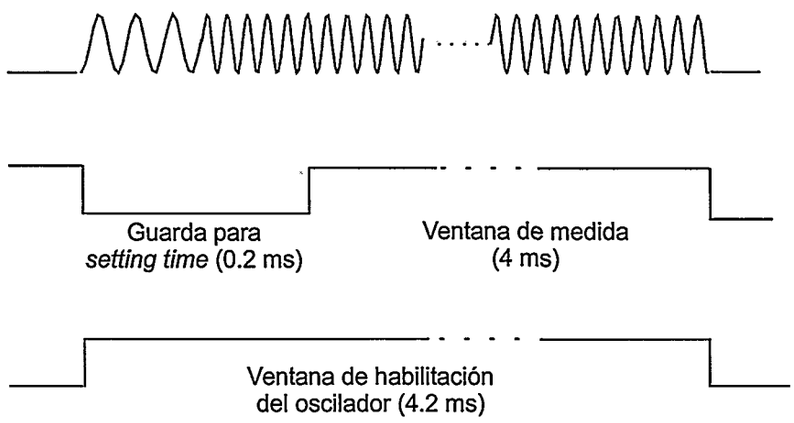



Fig. 43: Minimización de autocalentamiento: esquema de habilitación del oscilador

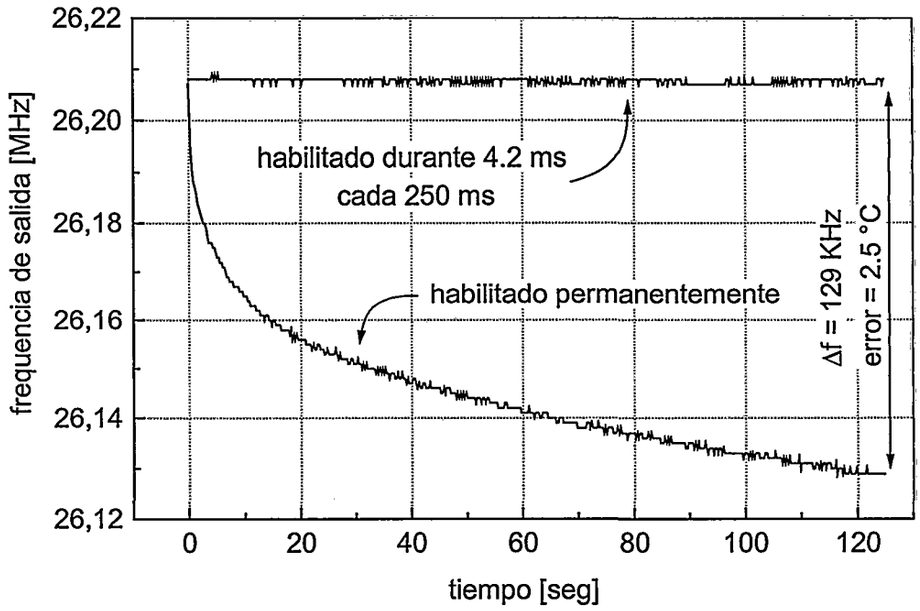



Fig. 44: Efecto de autocalentamiento en función del tiempo de habilitación del sensor

# 4. Resultados experimentales

De la Fig. 45 hasta la Fig. 60 se resumen los principales resultados experimentales para los circuitos en FPGAs de la familia XC4000, mientras que en la Fig. 61 hasta la Fig. 64 se muestran los primeros experimentos, realizados sobre la serie XC3000. A continuación, desde la Fig. 65 hasta la Fig. 68 se comparan las respuestas obtenidos en dispositivos de diferentes familias. Por último, a partir de la Fig. 69 se presentan las medidas de los diodos de enclavamiento. Todas los experimentos han sido realizados en condiciones idénticas y sobre los mismos chips de prueba.

En la Fig. 45 y en la Fig. 46 se muestran las respuestas en temperatura de los circuitos mapeados en la FPGA XC4005EPC84-3C: Ring1 a Ring8, y OSC4. Como se puede ver, el comportamiento es claramente lineal en el rango normal de funcionamiento del circuito integrado. La mayor frecuencia de operación corresponde a Ring2, que es el circuito para el que la herramienta de análisis de tiempos había dado menores retardos, mientras que el resto de sensores permanece en la banda de 20 a 30 MHz. La excepción es OSC4, con frecuencias alrededor de 7 MHz. Como ya se ha comentado, una menor frecuencia de operación es recomendable, porque disminuye el autocalentamiento.

En lo referente a la matriz de osciladores idénticos, se puede observar en la Fig. 46 que sus salidas no están completamente apareadas en frecuencia absoluta. La máxima diferencia ronda alrededor de 0.8 MHz, equivalente al 3%, aunque dos de ellos (Ring5 y Ring6) tienen un comportamiento prácticamente idéntico. Este resultado es importante, porque indica que un layout idéntico no implica la misma frecuencia de operación.

Otro dato importante es que, como era de esperar, los datos proporcionados por la herramienta de análisis estático de tiempos sirven sólo para averiguar si un circuito oscilará más rápido que otro, pero no para predecir con exactitud la frecuencia de salida.

En la Fig. 47 y en la Fig. 48 se muestran las respuestas en temperatura en forma de variación porcentual. Lo que resulta más interesante de estas gráficas es que aunque los circuitos oscilan a frecuencias muy diferentes, su variación en tanto por ciento por grado es muy parecida en todos los casos: Ring3 es el que tiene la menor, 0,19%, sólo tres décimas menor que la de Ring1, que con 0,22% es el que tiene mayor variación. Estos resultados hacen factible que se pueda sustituir la calibración por una sencilla normalización: tomar sólo el valor de la frecuencia a temperatura ambiente y luego utilizar

el mismo coeficiente de variación para todos los sensores, p.ej. 0,20% por °C. Por lo visto en la figura, sólo se cometería un error de aprox. ±5 °C en un rango de más de 100 °C. La Fig. 48 es muy relevante, porque indica que a layouts iguales les corresponden iguales variaciones porcentuales. O sea, que sólo es necesario calibrar un único sensor; para el resto sólo con normalizarlos es suficiente, para compensar las diferencias en frecuencia absoluta. Los coeficientes de variación son tan parecidos que sólo se cometería un error menor de ±1 °C en el rango de 100 °C.

El aspecto más negativo de los osciladores en anillo es su sensibilidad frente a variaciones en Vcc; en la Fig. 49 a la Fig. 52 se muestran estas variaciones para los circuitos implementados en la XC4005EPC84-3C. Lo primero que se puede comprobar es la importancia de los errores a los que puede inducir esta sensibilidad: alteraciones de ±5% en la tensión de alimentación provocan errores típicos de ±15 °C en la lectura del sensor. En cualquier caso, y aunque estos errores son enormes, no son lo suficientemente grandes como para que enmascaren subidas peligrosas en la temperatura de operación, que serán de decenas de °C. Pero es evidente que si se quiere hacer una medida precisa de la temperatura, hay que habilitar algún mecanismo de compensación.

En cualquier caso, la respuesta frente a Vcc es muy parecida a la de la temperatura. Es también lineal, aunque no sea tan pura como para la temperatura; tiene un aspecto ligeramente parabólico. En este caso también ocurre que las variaciones porcentuales son muy similares en todos los sensores, desde el 11,6% de Ring1 al 12.6% de Ring3. Y otro dato importante, las variaciones porcentuales en la matriz de sensores son iguales, lo que una vez más ayuda a simplificar la calibración.

A continuación, de la Fig. 53 a la Fig. 56 se muestran los resultados de los circuitos mapeados en una FPGA de la misma familia pero más antigua, la XC4005PC84-6C. Probablemente lo más interesante de estas figuras es lo parecidas que son a las anteriores; esto demuestra que el comportamiento de los sensores no varía apreciablemente al cambiar de FPGA. Aunque las frecuencias de oscilación absolutas son diferentes porque la FPGA es más lenta y hay circuitos con layout diferente (Ring11), las variaciones porcentuales son muy similares: siguen estando en el orden de 0,21% por °C para la temperatura, y en el caso de Vcc son un poco mayores, llegando al 18,7% de Ring11.

Con respecto a Ring11, el oscilador construido sólo con un IOB, lo primero que hay que remarcar es su elevada frecuencia de operación, casi 43 MHz a 20 °C. Esto es debido a que es el que tiene el mínimo retardo de rutado: una pista de 10.8 ns para conectar internamente el `IBUF` con el `OBUF` (la conexión `OBUF-IBUF` está permanentemente habilitada en la mayoría de las FPGAs y PLDs). Y además, su retardo combinacional también es el menor, sólo 9 ns correspondientes a la suma del `IBUF` más el `OBUF`. Aunque el área ocupada es muy pequeña, no es un sensor recomendable porque su alta frecuencia de oscilación lo hace más propenso al autocalentamiento. Y además, su sensibilidad frente a variaciones de Vcc ha resultado ser la mayor, de 18,7% por voltio.

Por último, de la Fig. 57 a la Fig. 60 se muestra la comparación en las medidas de la célula interna OSC4 en las FPGAs XC4005EPC84-3C y XC4005PC84-6C. Para facilitar la comparación con los sensores basados en CLBs, se seleccionó para las medidas únicamente la salida de 8 MHz. Los resultados obtenidos son muy positivos; primero porque tienen la menor frecuencia de oscilación de todos los circuitos; y además, esta frecuencia podría ser todavía mucho menor si se hubiera escogido otra salida. Pero más importante todavía, porque de todos los sensores estos son los que tienen mayor sensibilidad respecto a la temperatura (0,24% y 0,28% por °C) y menor con respecto a Vcc (3,58% y 3,11% por voltio). Por todas estas razones la celda OSC4 es muy recomendable como transductor de temperatura.

Comparando el comportamiento en los dos dispositivos, puede verse que es bastante parecido; con la salvedad de la frecuencia absoluta, que es mucho menor en OSC4b por estar implementado en una FPGA más lenta. Las medidas de la variación porcentual de OSC4b con respecto a las variaciones de Vcc tienen un ajuste tan pobre porque se midieron con una precisión insuficiente, de decenas de KHz. Este fue un problema de los primeros experimentos, que fue posteriormente solucionado. Otro punto a remarcar es que la respuesta en temperatura tanto de OSC4 como de OSC4b no es tan lineal como en el caso de los osciladores en anillo construidos con los recursos de la FPGA; en este caso la salida tiene un aspecto ligeramente parabólico.

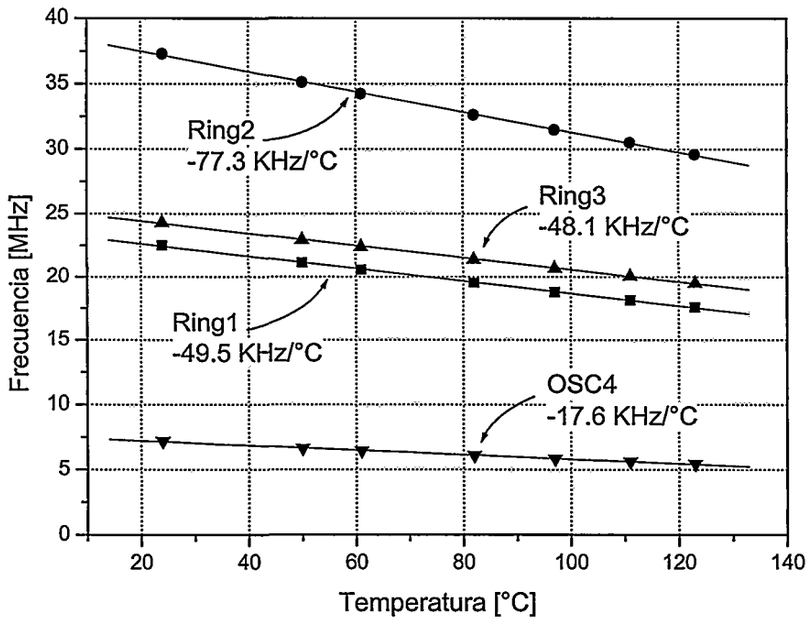



Fig. 45: Frecuencia de salida vs. temperatura. Ring1, Ring2, Ring3 y OSC4

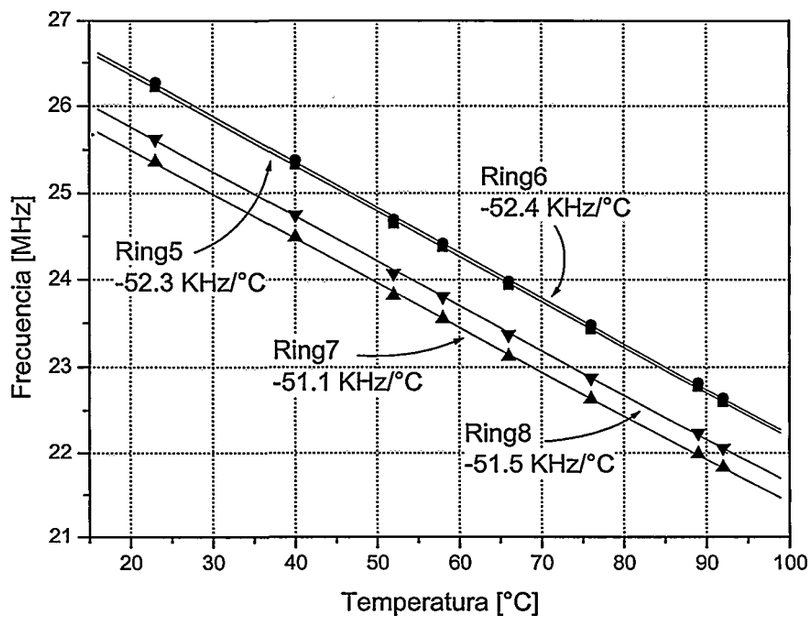



Fig. 46: Frecuencia de salida vs. temperatura. Ring5 a Ring8

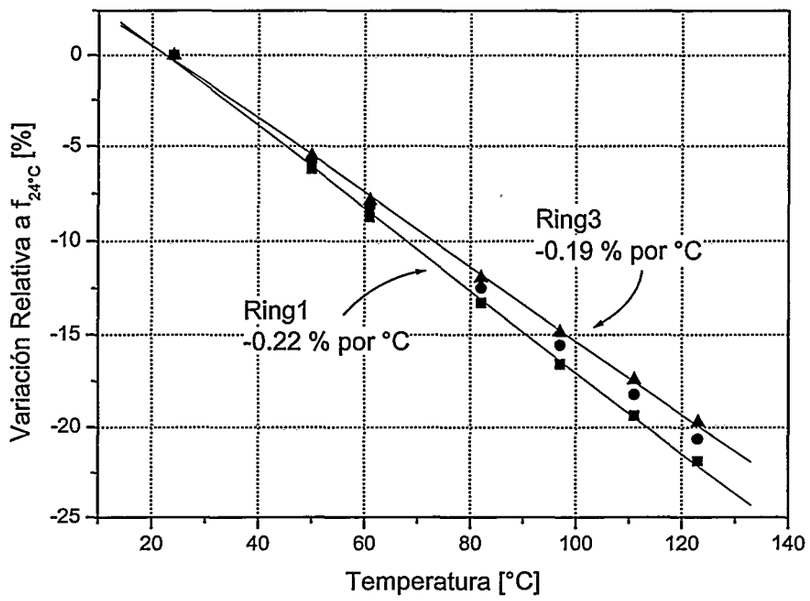



Fig. 47: Extremos de la variación porcentual de la respuesta vs. temperatura. Ring1 a Ring3

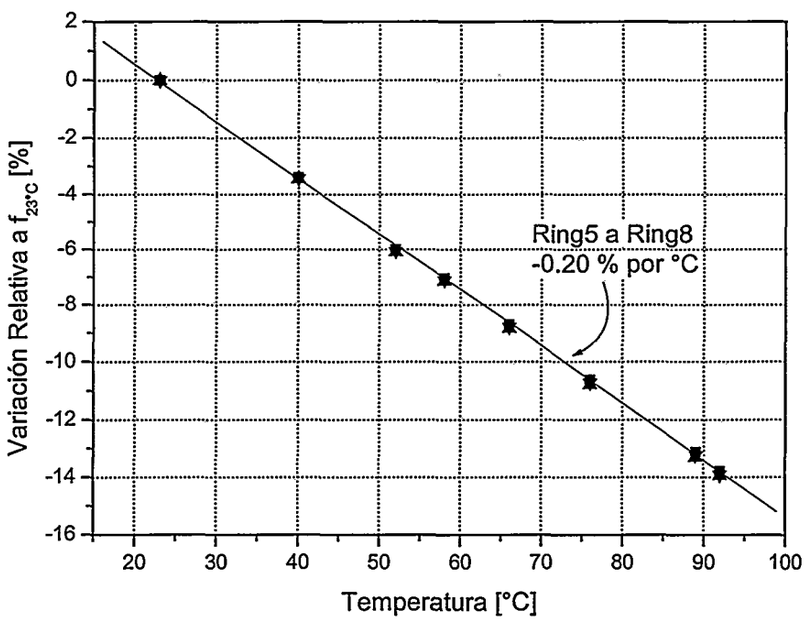



Fig. 48: Variación porcentual de la respuesta vs. temperatura. Ring5 a Ring8

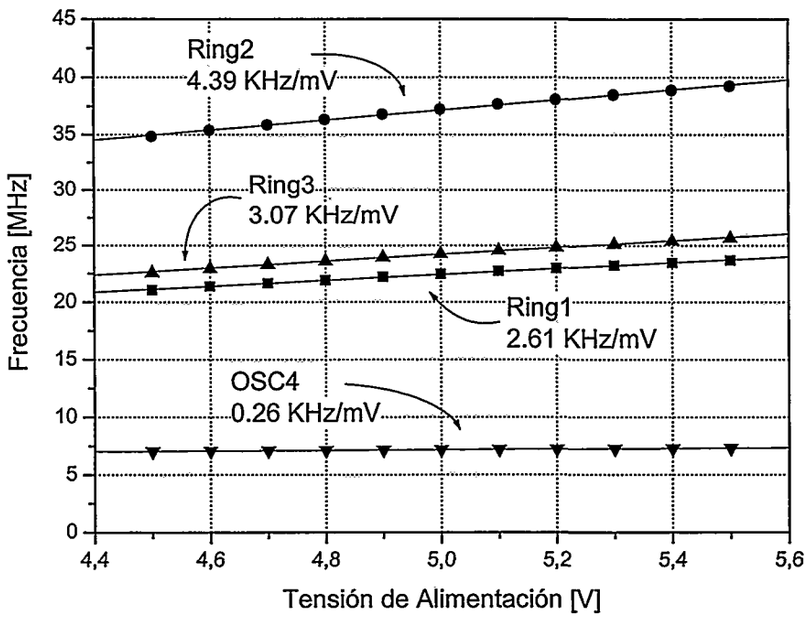



Fig. 49: Frecuencia de salida vs. Vcc. Ring1, Ring2, Ring3 y OSC4

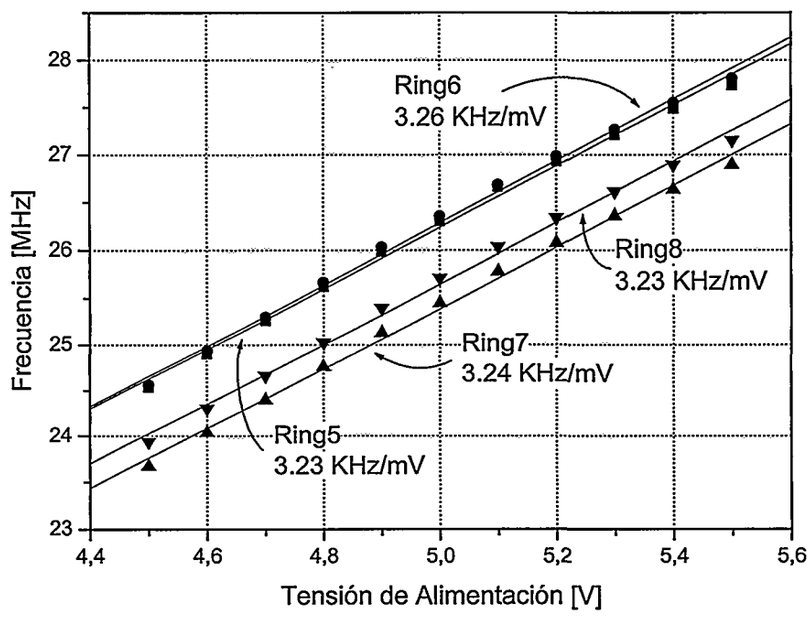



Fig. 50: Frecuencia de salida vs. Vcc. Ring5 a Ring8

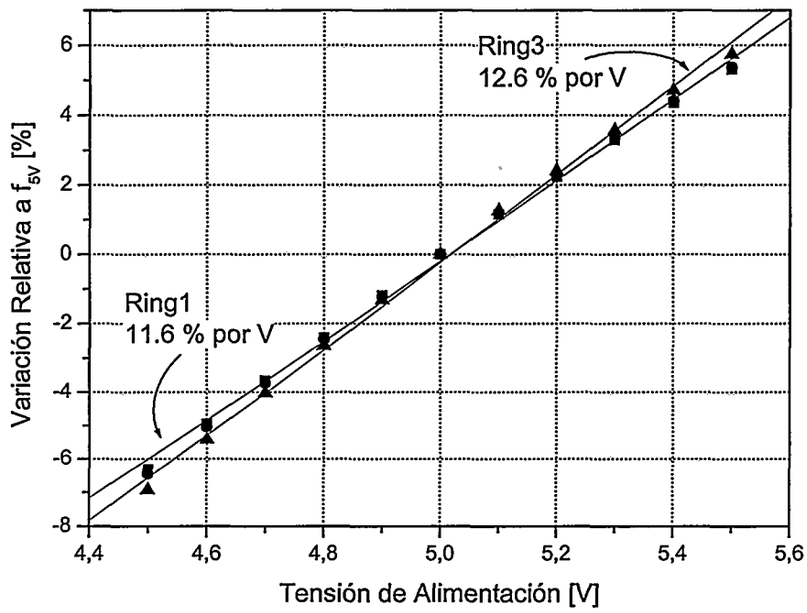



Fig. 51: Extremos de la variación porcentual de la respuesta vs. Vcc. Ring1 a Ring3

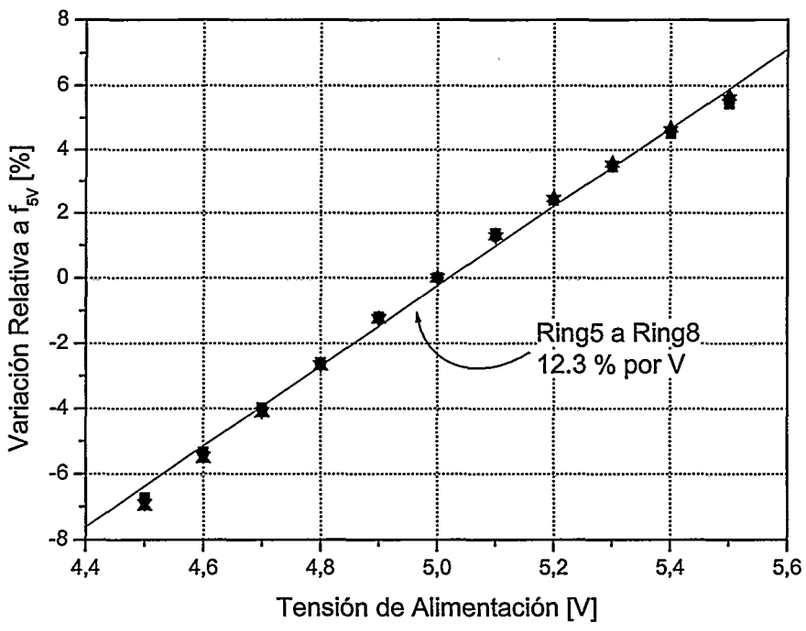



Fig. 52: Variación porcentual de la respuesta vs. Vcc. Ring5 a Ring8

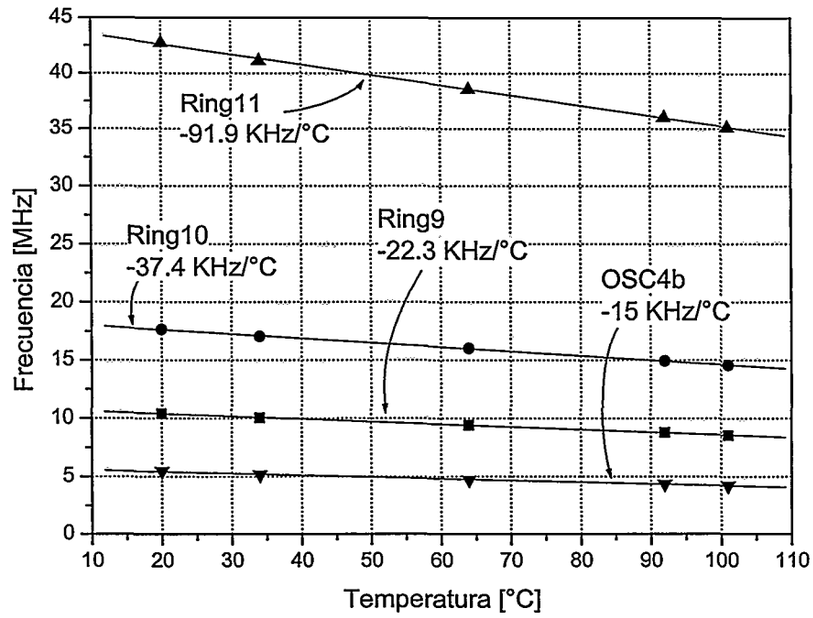



Fig. 53: Frecuencia de salida vs. temperatura. Ring9, Ring10, Ring12 y OSC4b

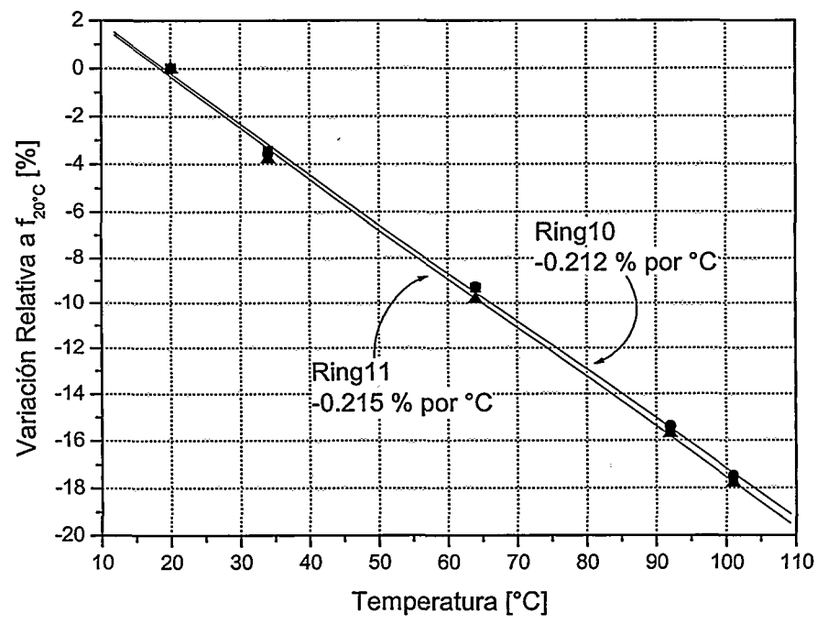



Fig. 54: Extremos de la variación porcentual de la respuesta vs. temperatura. Ring9 a Ring11

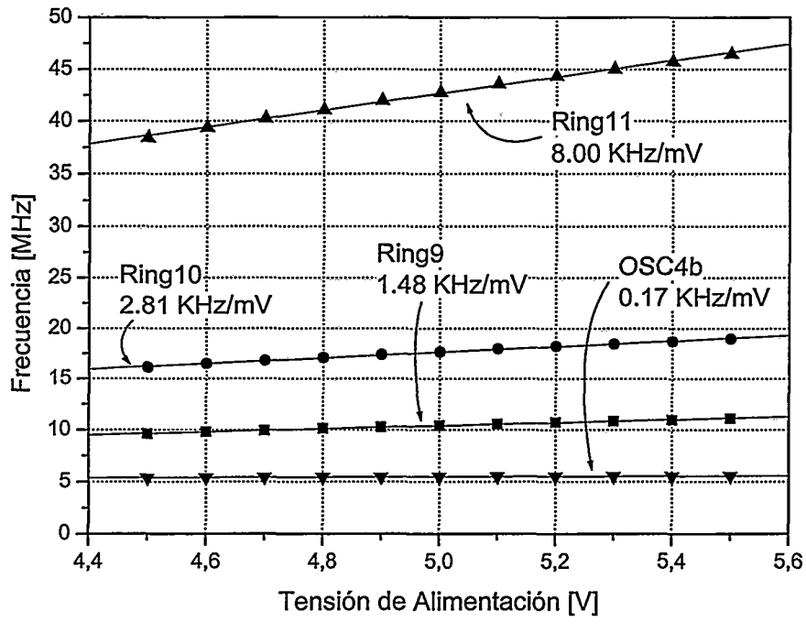



Fig. 55: Frecuencia de salida vs. Vcc. Ring9, Ring10, Ring11 y OSC4b

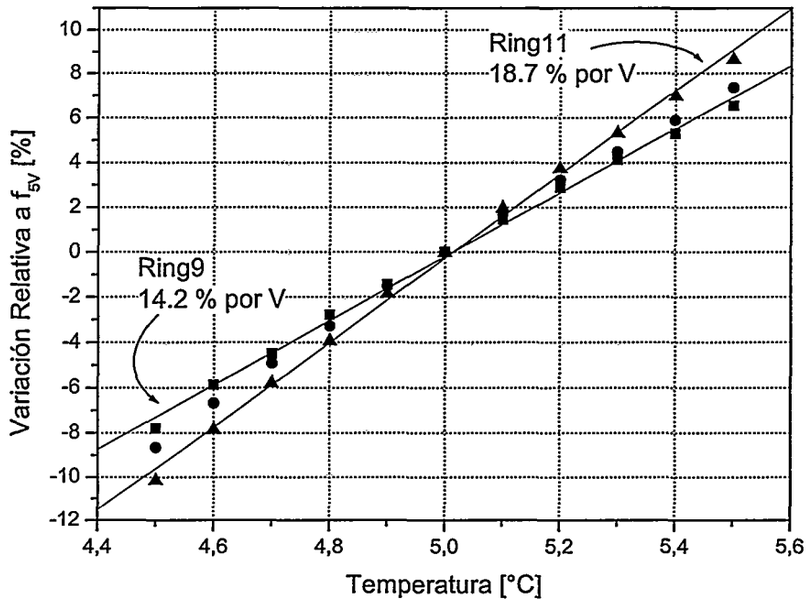



Fig. 56: Extremos de la variación porcentual de la respuesta vs. Vcc. Ring9 a Ring11

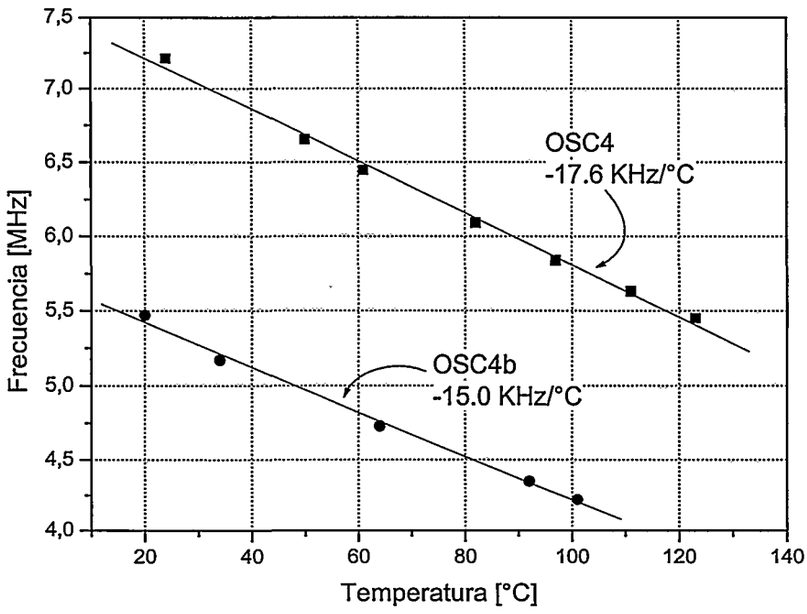



Fig. 57: Comparación en la respuesta vs. temperatura de OSC4 y OSC4b

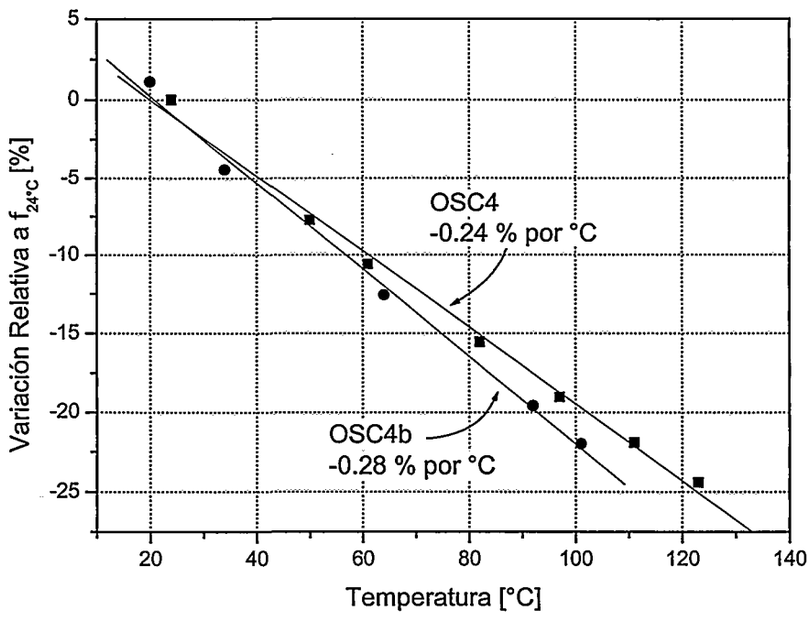



Fig. 58: Comparación en la variación porcentual vs. temperatura de OSC4 y OSC4b

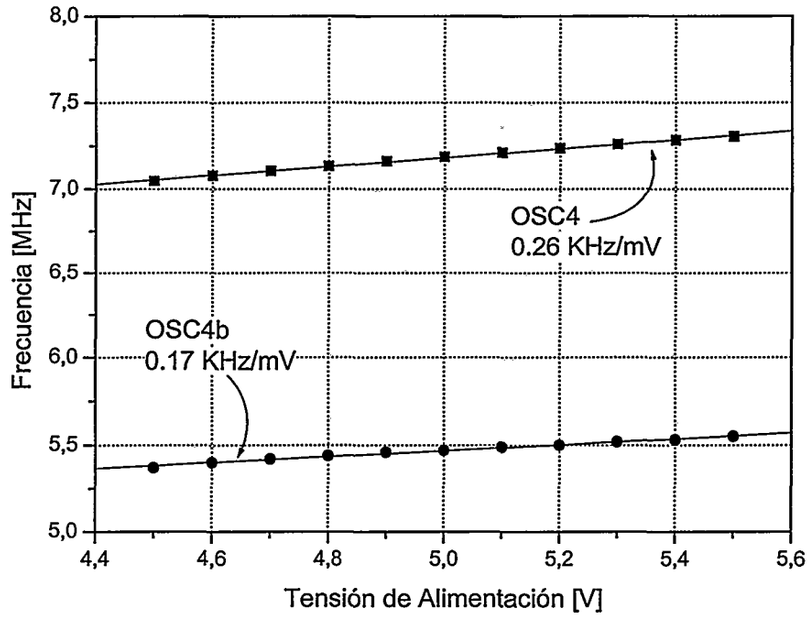



Fig. 59: Comparación en la respuesta vs. Vcc de OSC4 y OSC4b

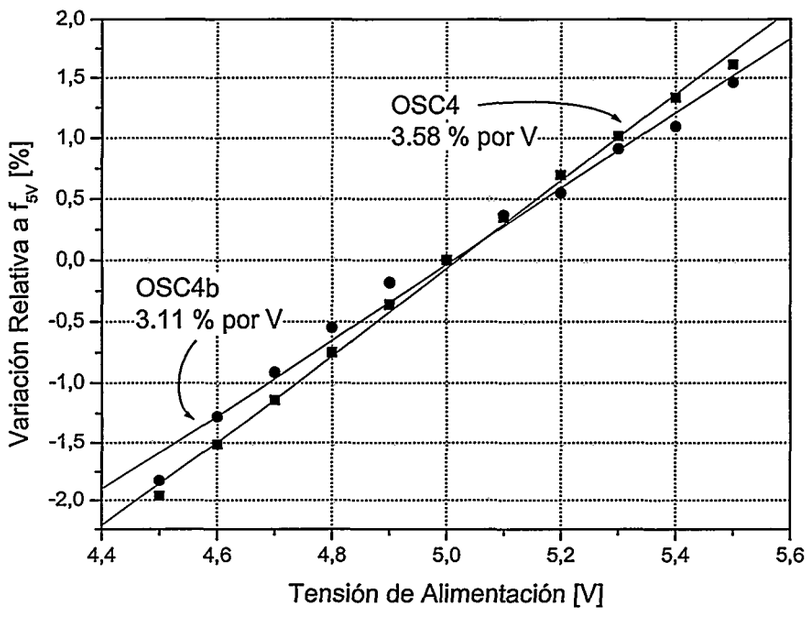



Fig. 60: Comparación en la variación porcentual vs. Vcc de OSC4 y OSC4b

A continuación, de la Fig. 61 a laFig. 64 se muestran los resultados para la FPGA más antigua de las utilizadas, la XC3030PC84-125C. Una vez más, lo primero que salta a la vista es lo parecido de estas gráficas a las de la familia XC4000. Aunque sólo tres dispositivos sea poco como para poder generalizar con seguridad, todo parece indicar que el comportamiento de los osciladores en anillo debe ser muy parecido en todas las FPGAs. En este caso en particular, las variaciones porcentuales con respecto a la temperatura han resultado ser un poco mayores, de entre 0,23% y 0,25% por °C, mientras que las sensibilidades respecto a variaciones de Vcc están en el rango de las medidas para la familia XC4000; van desde el 13,5% por voltio de Ring12 a el 14,8% de Ring13.

Una vez más, el circuito que oscila más rápido es el basado en un único IOB: Ring15, con una frecuencia de casi 45 MHz a temperatura ambiente (23 °C). Al igual que pasaba en la familia XC4000, esto es debido a que su retardo combinacional es pequeño (sólo un IBUF más un OBUF), y además, sólo tiene un nodo fuera del IOB, por lo que su retardo de rutado tampoco es grande. Como ya se indicó en este caso, la elevada frecuencia de operación de estos circuitos hace que no sean muy recomendables, incluso pese a su reducida área.

Seguidamente, de la Fig. 65 a la Fig. 68 se hacen comparativas entre las respuestas de circuitos implementados en distintos dispositivos. Primero para osciladores basados en CLBs, con similares retardos combinacionales y de rutado (Fig. 65 y Fig. 66), y más tarde para los basados en IOBs (Fig. 67 y Fig. 68). Como ya se ha venido viendo hasta ahora, la conclusión es que los osciladores en anillo se comportan de una manera muy parecida en todas las FPGAs utilizadas. Quizá lo más interesante de estas gráficas sea que la respuesta en temperatura para dos osciladores con el mismo layout, Ring2 y Ring10, es prácticamente igual: esto podría indicar que si mismos layouts significan iguales variaciones porcentuales, incluso para dispositivos diferentes. De confirmarse, esto sería sin duda muy útil para reducir el proceso de calibración a una simple normalización; independiente incluso de la FPGA, siempre que sea de la misma familia. Sin embargo, esta similitud en las sensibilidades no se mantiene para las variaciones de Vcc.

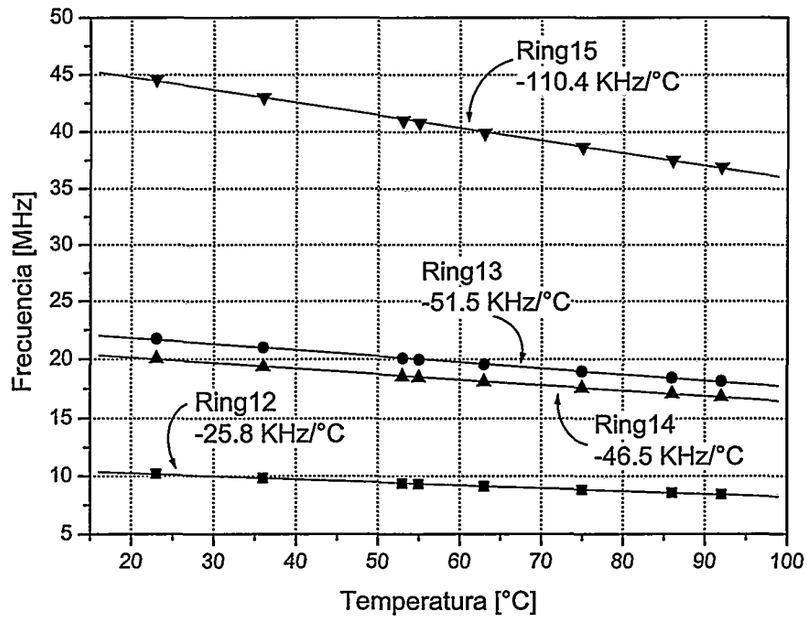



Fig. 61: Frecuencia de salida vs. temperatura. Ring 12 a Ring15

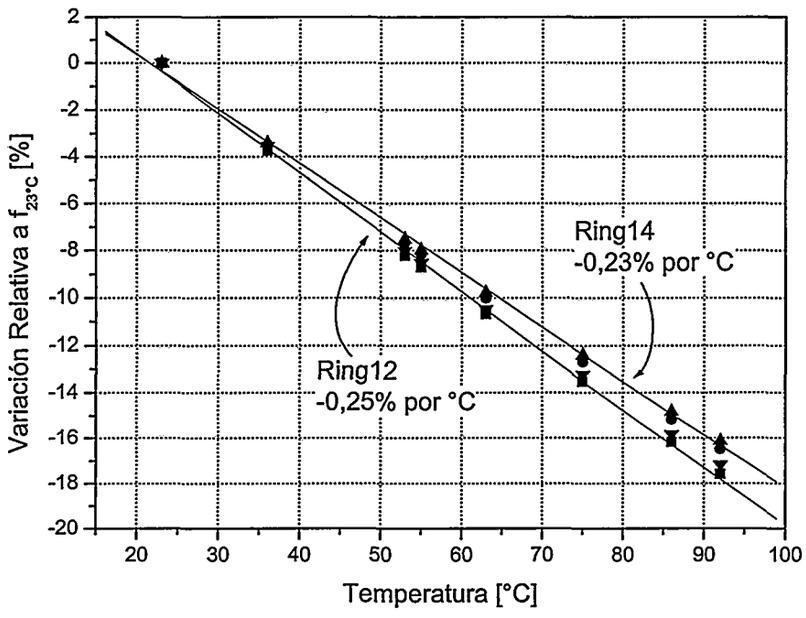



Fig. 62: Extremos de la variación porcentual de la respuesta vs. temperatura. Ring12 a Ring15

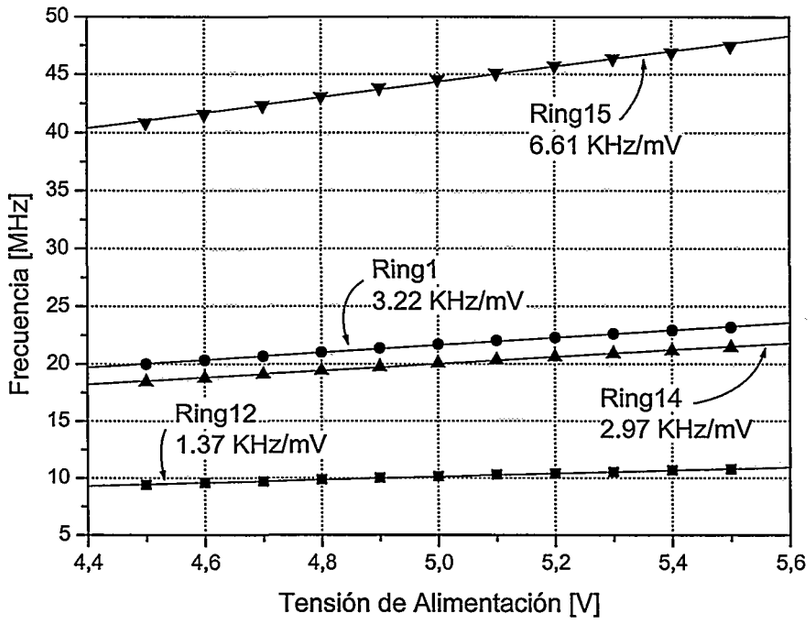



Fig. 63: Frecuencia de salida vs. Vcc. Ring12 a Ring15

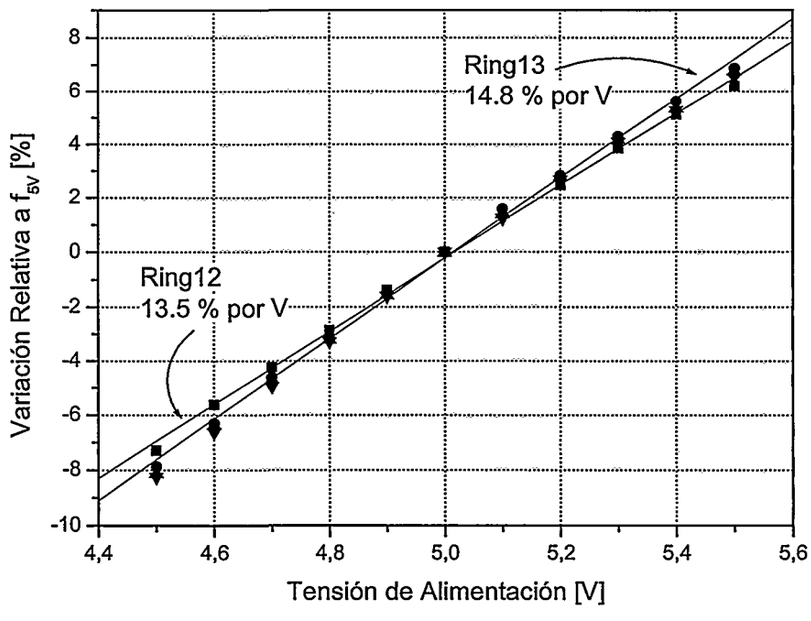



Fig. 64: Extremos de la variación porcentual de la respuesta vs. Vcc. Ring12 a Ring15

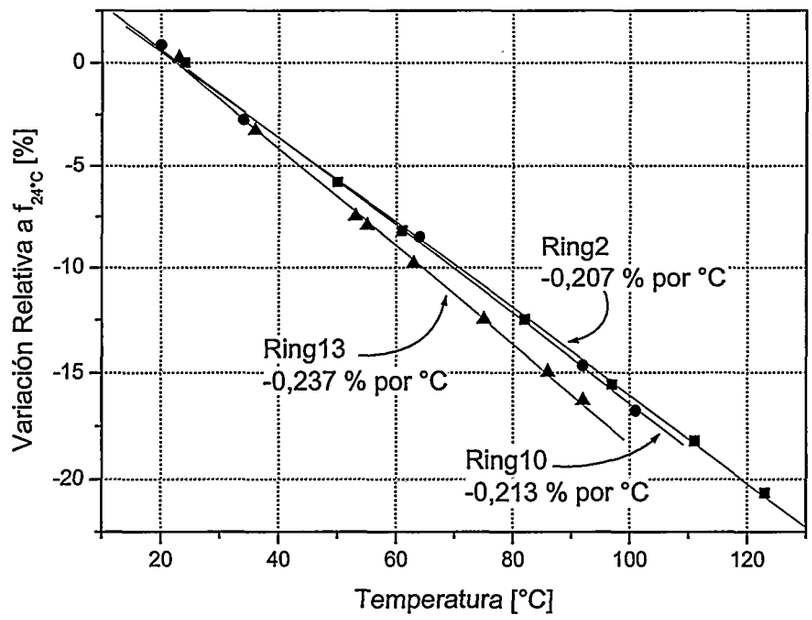



Fig. 65: Comparación en la variación porcentual vs. temperatura de Ring2, Ring10 y Ring13

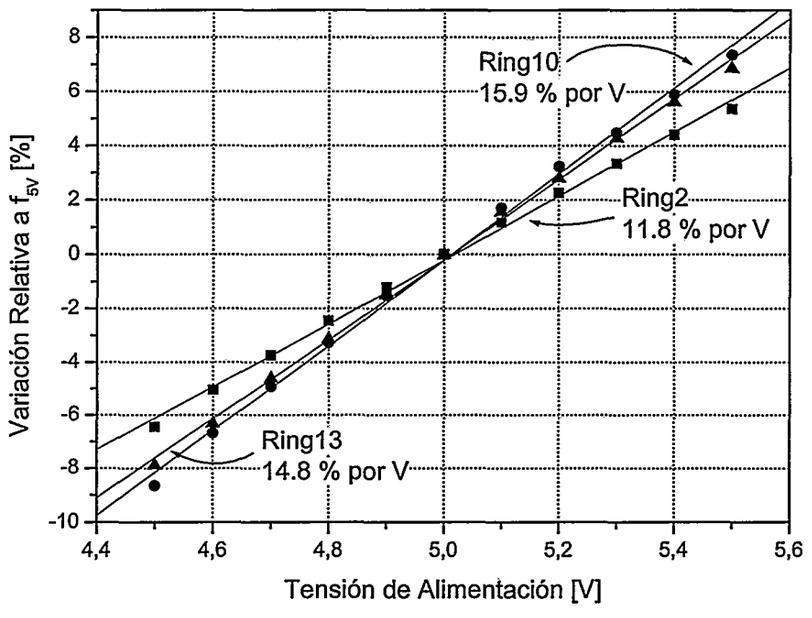



Fig. 66: Comparación en la variación porcentual vs. Vcc de Ring2, Ring10 y Ring13

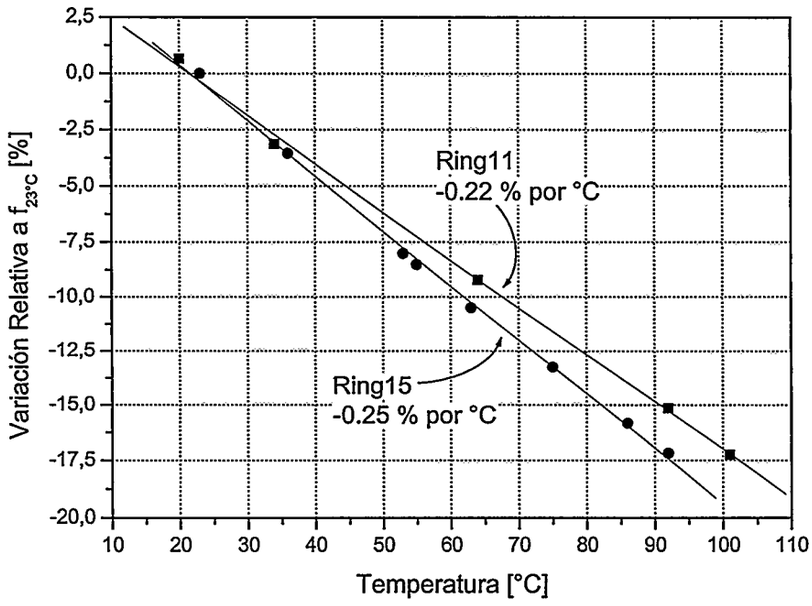



Fig. 67: Comparación en la variación porcentual vs. temperatura de Ring11 y Ring15

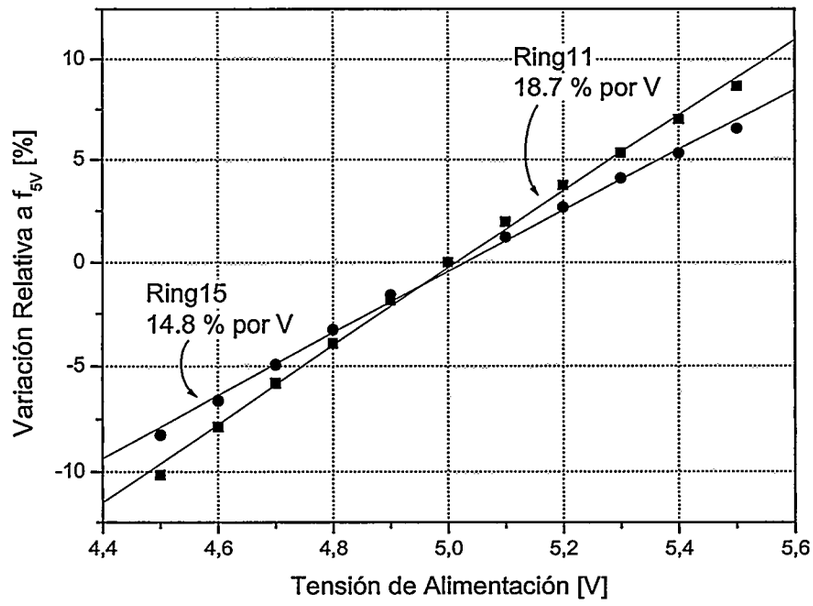



Fig. 68: Comparación en la variación porcentual vs. temperatura de Ring11 y Ring15

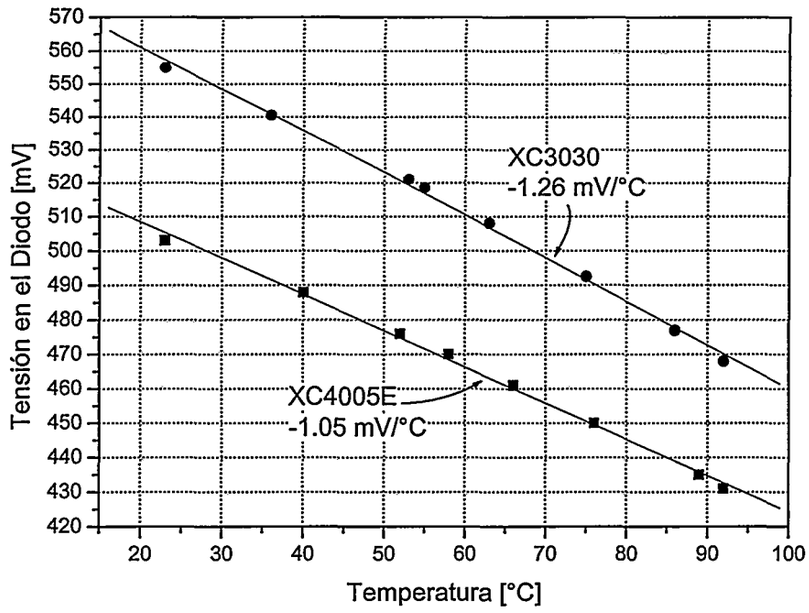



Fig. 69: Respuesta vs. temperatura de los diodos de enclavamiento. XC4005E y XC3030

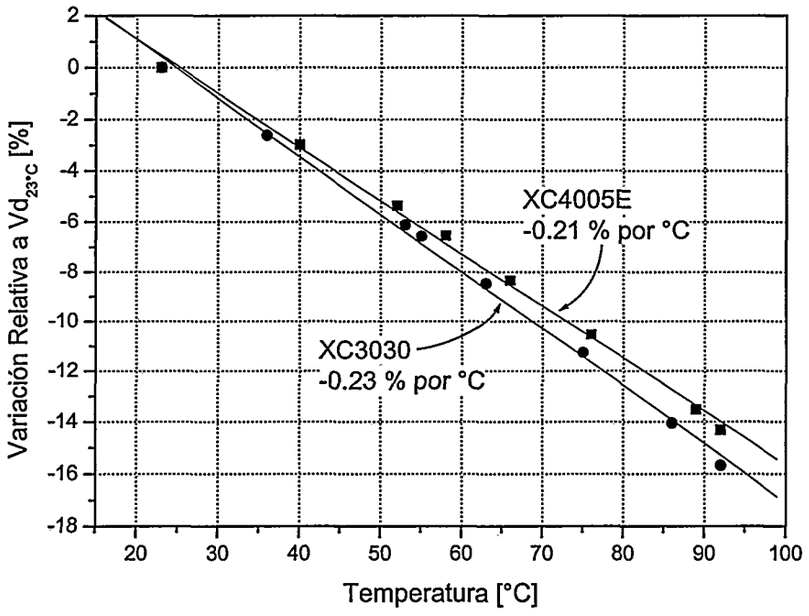



Fig. 70: Variación porcentual de la respuesta vs. temperatura de los diodos. XC4005E y XC3030

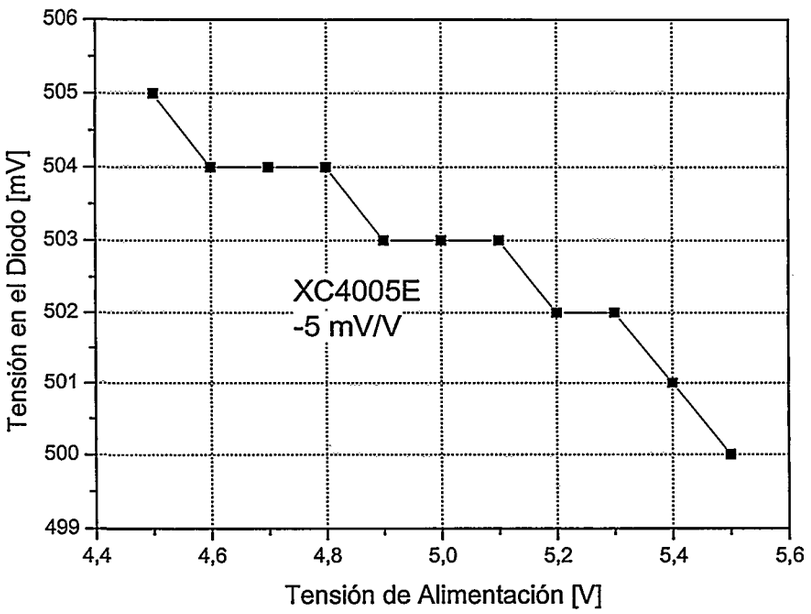



Fig. 71: Respuesta vs. Vcc del diodo de enclavamiento en XC4005E

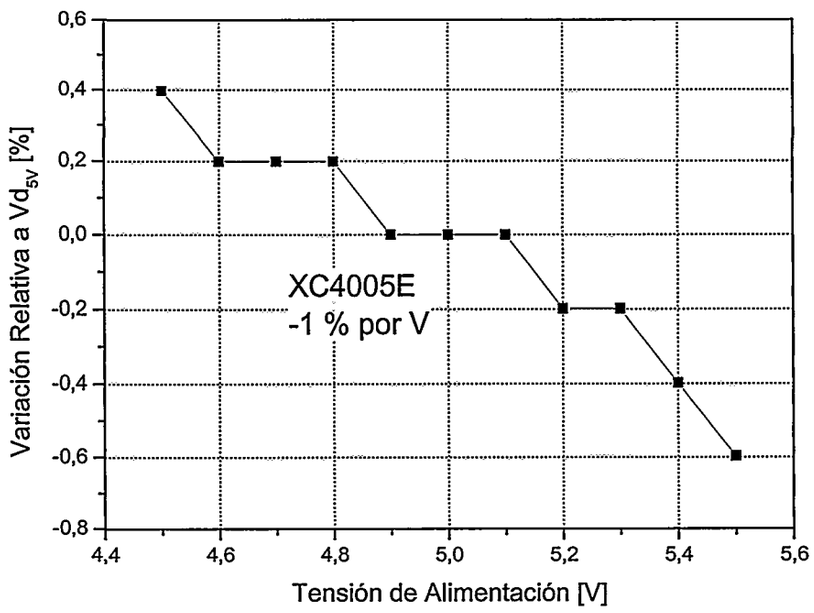



Fig. 72: Variación porcentual de la respuesta vs. Vcc del diodo de enclavamiento en XC4005E

Finalmente, de la Fig. 69 a la Fig. 72 se presentan los resultados de la última opción: el diodo de enclavamiento de los patas de la FPGA. Las medidas se realizaron con la fuente de corriente de 0,991 mA de la Fig. 41. Como se puede ver en las figuras, la respuesta no es tan lineal como en los osciladores en anillo, y la sensibilidad tampoco es alta: entre un 0.21 % y 0.23 % por °C, similar a la de los circuitos anteriores. Sin embargo, los diodos tienen una importante ventaja: teóricamente son insensibles a las variaciones en la tensión de alimentación. Aunque en esta caso si que se observó una pequeña sensibilidad, probablemente por la interferencia del resto de componentes del IOB, si que es cierto que es muchísimo menor que en los otros sensores. Mientras que los osciladores en anillo presentan una variación cercana a un 15% por voltio, los diodos tienen sensibilidades menores a un 1% (Fig. 72).

A continuación se muestra una figura que es el resumen de los resultados obtenidos para todos los circuitos anteriores. La mejor opción corresponde a la célula OSC4 de Xilinx: tiene una sensibilidad baja a Vcc, que se aproxima a la del diodo, junto con una sensibilidad alta respecto a la temperatura, similar a la de los osciladores en anillo.

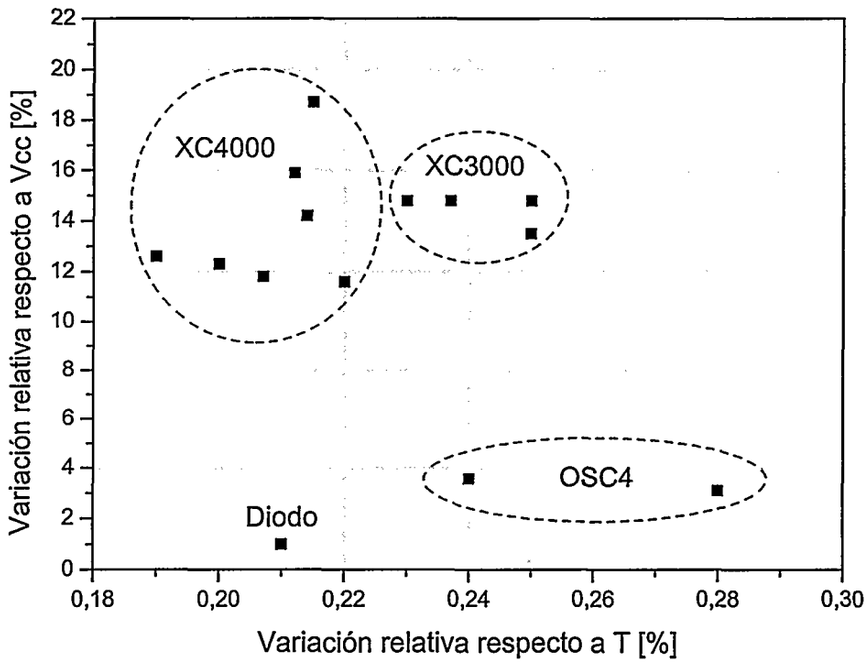



Fig. 73: Resumen de resultados

# 5. Experimentos sobre colisiones en pines y buses

Para ilustrar la sensibilidad de los sensores desarrollados en este capítulo, se utilizó la matriz de osciladores Ring5 a Ring8 para detectar colisiones en pines de salida y en buses internos.

En el primer caso, se fijaron las patas 35 y 36 de una FPGA XC4005 a niveles lógicos opuestos. Ambos terminales están situados en la esquina inferior derecha, próximos al circuito Ring6. El experimento consistió en producir un cortocircuito controlado entre ambos pines y monitorizar la respuesta del array. Los resultados se resumen en la Fig. 74, que muestra la frecuencia normalizada de los sensores respecto a sus valores a temperatura ambiente. En t=0, se produce el cortocircuito y termina en t = 25s. Un análisis de los datos indica que todas las frecuencias disminuyen con una exponencial de segundo orden. Sin embargo, el decaimiento es diferente para cada sensor. Un aumento de temperatura en la esquina del chip es descubierta por el Ring6: cerca de 3°C superior al valor correspondiente a la esquina opuesta. La temperatura en las esquinas equidistantes (Ring5 y Ring7) resultan casi iguales.

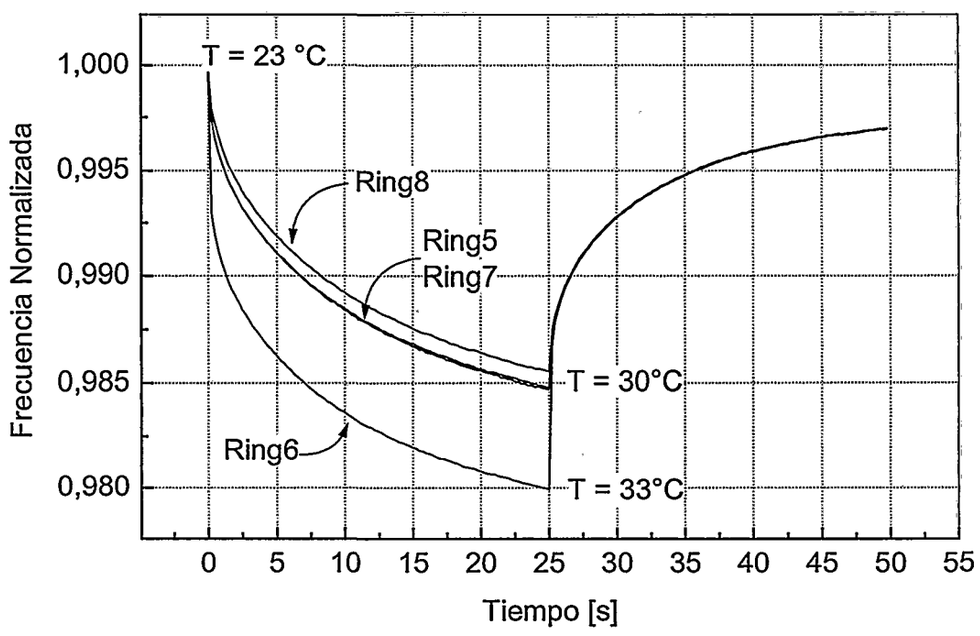



Fig. 74: Efecto en la respuesta de los sensores producido por una colisión entre pines de salida

El experimento anterior fue repetido para un cortocircuito entre dos búferes internos de tercer estado (TBUF) situados cerca de Ring6. Incluso considerado las pequeñas magnitudes involucradas en este caso, la respuesta que se obtiene es similar a la hallada en la prueba anterior (

Fig. 75). El experimento también muestra que los sensores pueden utilizarse para determinar si existe un gradiente de temperatura en el dado durante el funcionamiento normal del circuito.

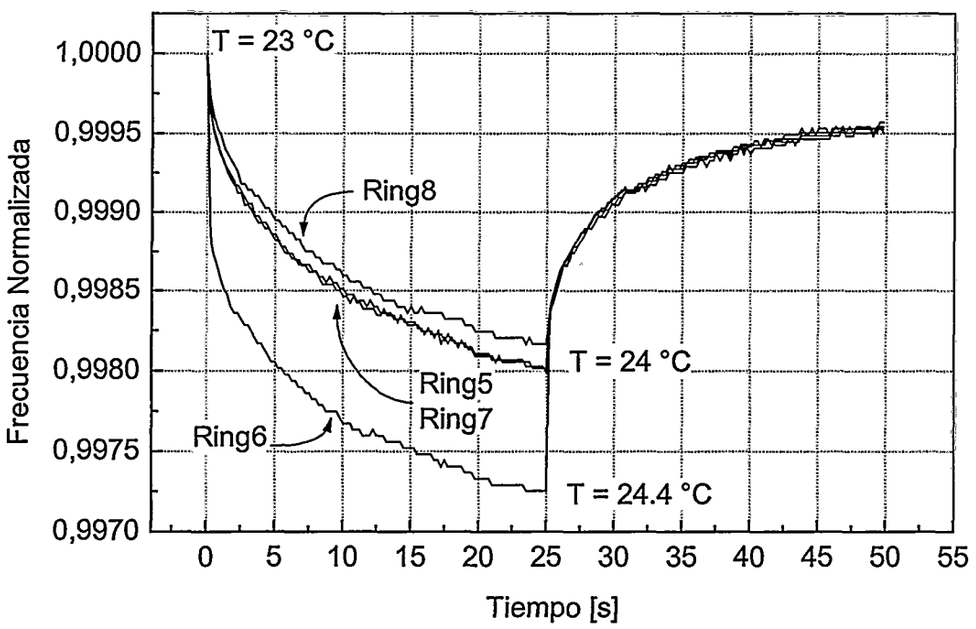



Fig. 75: Efecto en la respuesta de los sensores producido por una colisión en un bus interno

# 6. Conclusiones

En este capítulo se han presentado los métodos que se han desarrollado en esta tesis para medir la temperatura del silicio en dispositivos programables, y se ha demostrado su validez en una serie de experimentos, realizados en diferentes familias de FPGAs.

Sin duda alguna, la técnica más interesante es utilizar osciladores en anillo construidos con los propios recursos programables de la FPGA, por las siguientes razones:

- Son circuitos completamente digitales, que no necesitan de ningún componente analógico adicional. Además, al ser digitales son menos sensibles a los problemas de ruido.
- Son muy sencillos: nada más que un numero impar de inversores, por lo que se pueden construir con prácticamente cualquier recurso de cualquier FPGA, y además, ocupan muy poca área.
- Tienen una buena sensibilidad, alrededor de 0,20% por °C.
- Los experimentos muestran que su comportamiento es muy parecido en muy variadas familias de FPGAs.
- Se pueden situar en cualquier punto del chip, lo que los convierte en una herramienta muy poderosa para estudiar aspectos térmicos de circuitos integrados. Por ejemplo, se podrían emplear para construir un mapa térmico de la FPGA en funcionamiento.

Sin embargo, uno de los puntos débiles de los osciladores en anillo es su fuerte sensibilidad frente a variaciones en Vcc. La técnica clásica del diodo embebido en el circuito elimina este problema, pero pocas familias de FPGAs lo incluyen, como se pudo ver en el capítulo anterior. Sin embargo, en este capítulo se ha demostrado que los diodos de enclavamiento disponibles en las patas de las FPGAs se pueden emplear como transductores de temperatura. Este método permite obtener sensores prácticamente insensibles a Vcc en cualquier dispositivo programable; el único precio a pagar es que es necesario añadir unos pocos componentes analógicos externos.

Por otro lado, también se ha descubierto que los osciladores dedicados como la celda OSC4 de la familia XC4000 de Xilinx pueden ser unos excelentes sensores de temperatura, pues aúnan una buena sensibilidad frente a la temperatura, de alrededor de 0,25% por °C, con una baja sensibilidad a las alteraciones de Vcc, de algo más de 3% por voltio.

Un punto que ha resultado de especial interés en los experimentos es utilizar matrices de sensores construidos a base de osciladores en anillo con el mismo layout.

- Los experimentos muestran que la variación porcentual de los sensores con el mismo layout es la misma, incluso en dispositivos de diferente tipo. Aunque la

frecuencia absoluta de oscilación varía de un circuito a otro, gracias a esta propiedad no es necesario calibrar todos los sensores, sino que es posible calibrar sólo uno, y los datos para los demás se pueden obtener normalizándolos.

- Disponer de varios sensores en un chip puede ser una herramienta muy potente para encontrar fallos; por ejemplo, en los experimentos se ha demostrado que es posible detectar colisiones y averiguar en que zona de la FPGA se han producido.

# Capítulo 5.

# Verificación térmica en arquitecturas reconfigurables en tiempo de ejecución

La idea principal de este capítulo es que la medida de la temperatura en FPGAs es una aplicación muy adecuada para ser manejada con reconfiguración en tiempo de ejecución. Para demostrar que esto es posible, se ha diseñado un sensor que puede ser dinámicamente insertado y eliminado de la FPGA, utilizando la herramienta JBits. Para medir la temperatura, este sensor utiliza una de las técnicas presentadas en el capítulo anterior: los osciladores en anillo. A parte de detallar su construcción, se presentan los resultados experimentales que demuestran su validez.

## 1. Definiciones previas

A modo de definición no formal, una arquitectura dinámicamente reconfigurable es aquella en la que existen dispositivos programables cuya configuración puede ser variada dinámicamente a lo largo del tiempo. Así, una FPGA que se reprograma por un microprocesador puede considerarse un sistema dinámicamente reconfigurable. Sin embargo, una FPGA que se configura a partir de una EPROM serie no lo es. Reconfiguración dinámica sólo implica la posibilidad de que haya cambios en la funcionalidad, aunque para realizarlos sea necesario hacer una pausa en la ejecución del

sistema. Dicho de otra manera, no se exige la operación en tiempo real. Pero si se garantiza la operación en tiempo real durante estos cambios en la configuración, entonces se obtiene la versión más eficaz de estas arquitecturas: las reconfigurables en tiempo de ejecución

## 2. Justificación del uso de reconfiguración

El sensado de temperatura de una arquitectura electrónica se realiza usualmente para verificar si se está trabajando dentro los márgenes seguros de operación. Pero también puede ser útil para evaluar cual es el bloque del circuito está provocando un incremento excesivo de la temperatura. Estos incrementos estarán causados bien por un error de diseño, o bien porque el bloque está operando con una carga capacitiva y velocidad excesiva.

Para el primer caso, verificar si se está trabajando dentro los márgenes seguros de operación, resulta suficiente un único sensor con una frecuencia relativamente baja (unas pocas medidas por segundo). No tiene sentido usar una matriz, pues los gradientes de temperatura (unos 5 °C, como mucho) no tienen relevancia. Tampoco tiene sentido medir con mayor cadencia, debido a las altas constantes térmicas del sistema.

Por el contrario, cuando se desea investigar la causa de una temperatura peligrosamente alta, si tendrá sentido hacer un sensado en varios puntos del dispositivo. La idea será obtener un mapa térmico que informe si el incremento de temperatura está provocado por un elemento concreto, o está localizado en una determinada área. Con estos datos, el diseñador podrá fijar el problema.

Como conclusión, en las aplicaciones más comunes los sensores deben estar activos sólo una fracción del tiempo muy pequeña, debiendo permanecer inactivos el resto del tiempo, para evitar autocalentamiento. Además, de todos los sensores que potencialmente pueden usarse, normalmente sólo uno será útil. Sólo si se detectan problemas se deberán activar el resto, para tratar de recopilar la información necesaria para solucionarlo.

Sólo en caso de necesitarse una verificación térmica detallada, será útil utilizar toda la potencia que proporciona la reconfiguración en tiempo de ejecución. El esquema sería el siguiente: un contexto hardware (una matriz de sensores) sólo opera en unos instantes

concretos, pudiendo (o debiendo) ser eliminados cuando no son necesarios. Su número y localización puede variar dinámicamente (se incrementará si hay una situación de alarma, con el objeto de construir el mapa térmico). En esta aplicación, las constantes térmicas del circuito juegan a favor de la RTR, al ser mucho mayores que el tiempo de reconfiguración. Aunque se cambiase todo el contenido de la FPGA por una matriz de sensores y desaparecieran los circuitos originales, todavía se mantendrían los gradientes de temperatura el tiempo necesario para medirlos.

Obviamente, todo esto es también posible realizarlo sin reconfiguración; en teoría es siempre posible pasar de RTR a un sistema estático, sólo hay que dejar previstas todas las posibles configuraciones en "tiempo de diseño". En este caso, habría que dejar permanentemente en la configuración tantos sensores como fuesen necesarios para realizar el mapa térmico más detallado, el que se utilizaría en el peor de los escenarios posibles. Además, habría que habilitar mecanismos para inhibir la actividad en los sensores cuando no estuvieran tomando ninguna medida, para eliminar el autocalentamiento. Como se ve, la opción estática es posible, pero implica un gasto de área muy grande. Un gasto que se puede eliminar empleando RTR.

Por último, no sólo es cierta la afirmación de que la monitorización de temperatura es un buena aplicación para RTR, sino que lo contrario también es válido: las aplicaciones que usen reconfiguración dinámica en tiempo de ejecución son buenas candidatas para el chequeo térmico. Estas arquitecturas serán especialmente propensas a sufrir dos problemas: colisiones entre dos salidas, y configuraciones erróneas de la FPGA. En ambos casos, el resultado será una disipación excesiva de potencia. Adicionalmente, existen dos factores adicionales a favor de la verificación térmica de sistemas RTR. Por un lado, las herramientas de chequeo de reglas de diseño que proporcionan algunos fabricantes de FPGAs sólo sirven para diseño estático, existiendo sólo unas pocas a nivel de investigación que hacen esta comprobación en diseños dinámicos [Rob00]. Por otra parte, debido a las propias características de la reconfiguración dinámica no se puede saber a priori que combinación de circuitos va a constituir la carga de una FPGA.

# 3. Caso de estudio RTR: familia Virtex y JBits

La reconfiguración dinámica ha sido en los últimos años un muy activo campo de investigación en FPGAs. Sin embargo, y aunque la tecnología en la que se basan estos dispositivos permita fácilmente este modo de operación, pocas han sido las iniciativas con éxito para poner en el mercado una FPGA que la soporte. Realmente, sólo ha habido una familia comercial de éxito: la Virtex de Xilinx. Otras alternativas han alcanzado una pobre cuota de mercado [Atm99, Atm03], o han fracasado [Xil97] o incluso ni se han llegado a comercializar [Fau97]

Las FPGAs de familia Virtex no tienen un bitstream continuo, sino que está organizado en *frames*; cada *frame* se corresponde con la configuración de una "rebanada" vertical del chip. Cuando se configura la FPGA no hace falta cargar siempre todo el dispositivo, sino que se pueden reconfigurar uno o varios *frames*: este es el mecanismo escogido para implementar la reconfiguración parcial. La configuración de estas FPGAs tiene además una ventaja adicional: se garantiza que si se reconfigura un *frame*, y hay bits de configuración que no cambian, no se van a producir glitches en esos bits. Esto, junto con el hecho de que se puede deshabilitar la activación de la señal de reset global con cada reconfiguración, abre las puertas a la reconfiguración en tiempo de ejecución.

A continuación se hará una breve descripción de los detalles técnicos de la configuración en Virtex, para que se pueda comprender mejor los experimentos que se van a presentar a final de este capítulo. La información detallada se puede encontrar en [Xil99c, Xil02d, Xil02e, Xil02f, Xil03]

Como ya se ha explicado, los *frames* (elemento mínimo que puede ser reconfigurado) se corresponden con rebanadas verticales de la FPGA. Así, 48 *frames* se define la configuración de una columna de CLBs, 54 los de una de IOBs, 64 los contenidos de una columna de BlockRAMs... Todo esto se detalla en la [Xil03]. El tamaño de estos *frames* no es constante, sino que los dispositivos más grandes tienen *frames* mayores. Esta información está resumida en la Tabla 10, donde se ha relacionado con la velocidad máxima de configuración (50 $MBs^{-1}$) para mostrar el tiempo mínimo necesario para hacer un cambio en uno de estos elementos de la FPGA.

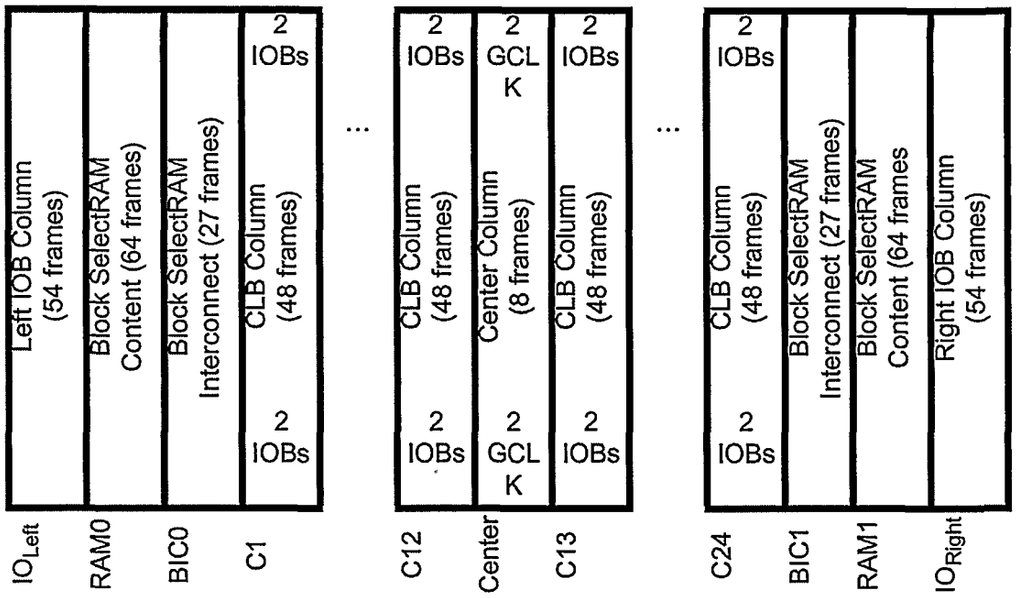



Fig. 76: Organización de los *frames* de configuración en Virtex (obtenida de [Xil03])

| Dispositivo | Bits por *frame* | Tiempo mínimo para reconfigurar [$\mu s$] | | |
|---|---|---|---|---|
| | | 1 *frame* | 1 LUT | 1 columna de CLBs |
| XCV50 | 384 | 0,96 | 15,36 | 46,08 |
| XCV300 | 672 | 1,68 | 26,88 | 80,64 |
| XCV800 | 1088 | 2,72 | 43,52 | 130,56 |
| XCV1000 | 1248 | 3,12 | 49,92 | 149,76 |

Tabla 10: Tiempos mínimos de reconfiguración en Virtex

En Virtex, la configuración puede realizarse bien través de la clásica interfaz serie (probablemente la más usada en FPGAs por lo fáciles que son de manejar las EPROMs serie), o usando una interfaz paralelo de 8 bits (SelectMap), o a través de JTAG. De todas estas alternativas, sólo las dos últimas permiten reconfiguración parcial; y además, también permiten readback. El readback en las FPGAs de Xilinx es una herramienta muy

potente, pues no sólo permite saber si la FPGA se ha configurado correctamente, sino que se puede usar para conocer el estado de todos los flip-flops y memorias del chip.

Por otro lado, la reconfiguración en tiempo de ejecución no es sólo cuestión del hardware, sino que deben existir herramientas que permitan manejarla. Desafortunadamente las ofertas en este campo son muy pocas, limitándose a unas pocas herramientas de investigación. A la hora de implementar los experimentos que se presentan en esta tesis se optó por emplear JBits, pues es la herramienta de soporte para RTR que más relevancia ha tenido hasta la fecha.

JBits es una herramienta suficientemente compleja como para que su descripción haya necesitado un apéndice de esta tesis; pero si se quisiera hacer una descripción muy concisa se podría decir que es un API que permite describir circuitos en Java. Desde un punto de vista teórico, su gran ventaja es que el diseño de circuitos pasa de ser algo estático a convertirse algo tan dinámico como la ejecución de un programa Java. De esta manera, el circuito se genera en tiempo de ejecución, según las necesidades de cada momento, y se carga dinámicamente en la FPGA. El diseño de circuitos deja de ser algo estático, donde las decisiones se toman en tiempo de diseño, para pasar a ser algo dinámico, donde los circuitos se crean según las condiciones actuales (funcionalidad requerida, espacio disponible en la FPGA, etc...) y las decisiones se toman en tiempo de ejecución. Dejando a parte las implicaciones de este cambio de metodología, que se discuten en el apéndice, la razón primordial para el uso de JBits en esta tesis es lo que facilita el proceso de reconfiguración. Con unas pocas líneas de código es posible cargar una nueva configuración, hacer un readback y volver a hacer una configuración parcial. Una vez más, todos los detalles y programas de ejemplo se presentan en el apéndice A.

## 4. Elección del tipo de sensor

Siguiendo las conclusiones del punto 2, el sensor ideal será aquel que se pueda añadir en tiempo de ejecución en cualquier posición de la FPGA, y que de la misma manera, sea posible eliminarlo una vez que haya realizado la medida, para dejar espacio a nuevos circuitos. Deberá ser por lo tanto un sensor totalmente autónomo y autocontenido.

En el capítulo anterior de esta tesis se han mostrado diversos tipos de sensores, cada uno con sus ventajas (y sus inconvenientes). Pero ninguno satisface este criterio de idealidad, y es que en todos ellos el sensado de temperatura es externo, por lo que

deben usar un circuito asociado y, al menos, un pad de entrada/salida. Y es precisamente este uso continuo de recursos lo que se trata de evitar. En cualquier caso, parece que esta limitación no es grave, pues el uso dedicado de pads de entrada/salida es algo poco más o menos que inevitable, porque los PCBs no son reconfigurables (por ahora). Aunque siempre se tuvieran que dejar uno o varios pads reservados para la conexión con el circuito de sensado de temperatura, el sensor todavía podría ser dinámicamente operado. Cuando se necesite, se agrega a la configuración de la FPGA, en la posición que se desee, y se crea dinámicamente un camino que rute su salida hasta los pads que van al circuito externo de sensado. Salvo que hubiera graves problemas de congestión en el rutado de la FPGA, esto no debería ser ningún problema.

Pero haciendo una valoración más detallada se comprueba que esta limitación puede ser más restrictiva de lo inicialmente pensado, y es que la reconfiguración para lo que es realmente interesante es para añadir un conjunto de sensores, y así obtener los mapas térmicos del dispositivo. Entonces, habría que disponer de tantos pads y circuitos externos de sensado como sensores de temperatura fuésemos capaces de añadir a la configuración actual de la FPGA. O implementar algún mecanismo de multiplexado, que no haría sino complicar más el problema. Parece claro que lo que hay que buscar es una alternativa que sea completamente autónoma, y que no use ningún recurso externo.

Por otro lado, una opción para realizar un mapa térmico del dispositivo sería ir 'moviendo' un único sensor por toda la FPGA. Esta opción sólo de necesitaría un único circuito de sensado y de un pad (o dos). Pero la limitación aquí es la velocidad con la que se debería hacer la reconfiguración. Aunque las constantes térmicas dominantes del sistema son grandes, si se multiplica el tiempo de reconfiguración por el número de sensores puede ser que llegue a un tiempo similar al de estas constantes, con lo que el método ya no sería válido.

Como conclusión, de todas las tres principales alternativas mostradas en el capítulo anterior, dos quedan inmediatamente descartadas. La primera son los diodos, porque necesitan siempre pads de entrada/salida y un circuito externo analógico. Y la segunda los osciladores dedicados, como OSC4, porque tienen una posición fija en la FPGA. La única opción que queda es construir osciladores en anillo con los recursos lógicos de la FPGA, que se usarían también para medir la frecuencia de salida (y por tanto, la temperatura). Esta opción si que es válida: en el capítulo anterior se ha demostrado que

es posible construir un oscilador en anillo en cualquier punto de la FPGA, y además, el circuito necesario para realizar el sensado puede ser construido en la FPGA sin mayor problema. Y además, en los siguientes puntos se demostrará que es posible construir un sensor completamente operado a través del puerto de configuración, con lo que queda satisfecho el último requerimiento que faltaba: que no utilizase pines adicionales de E/S.

# 5. Construcción de un sensor compatible con RTR

Como ya se ha visto, construir un sensor de temperatura con un oscilador en anillo es muy sencillo. Realmente sólo se necesitan tres elementos principales: el propio oscilador, un contador que mida la frecuencia de salida del oscilador, y otro contador que determine el tiempo durante el cual estará activo el oscilador. Al primer contador se le va a llamar de captura, y su entrada de reloj estará conectada a la salida del oscilador en anillo. El segundo contador se le denominará de base de tiempos, y estará conectado al reloj del sistema, que proporcionará una precisa referencia temporal. En la siguiente página, la Fig. 77 muestra el diagrama esquemático del sensor.

Para los contadores se ha escogido una longitud de palabra de 14 bits, como compromiso entre tamaño suficientemente pequeño pero que proporcionará una buena resolución para un amplio rango tanto de temperaturas (en el de captura) como de frecuencias del reloj del sistema (en el de base de tiempos). Para simplificar el diseño los dos contadores se han hecho iguales, lo que no ha resultado ser ningún problema para las frecuencias tanto de oscilación como de reloj del sistema empleadas en los experimentos.

El contador de base de tiempos fijará los intervalos durante los cuales se habilitarán tanto el oscilador en anillo como el contador encargado de medir su frecuencia. Al comenzar la cuenta se habilitará el oscilador en anillo, para que comience la oscilación. Una vez haya pasado el tiempo suficiente para que la oscilación se estabilice (en las observaciones directas se ha establecido que es unos pocos ciclos de reloj) se podrá habilitar el contador, que estará funcionando el tiempo necesario para poder tener una medida precisa de la frecuencia. Este tiempo se escogerá de tal manera que el contador en el peor de los casos (mínima temperatura) se quede cerca del desbordamiento. Una vez hecha la medición se deshabilitará toda la actividad en el sensor, incluyendo la cuenta de la base de tiempos, para evitar el autocalentamiento.

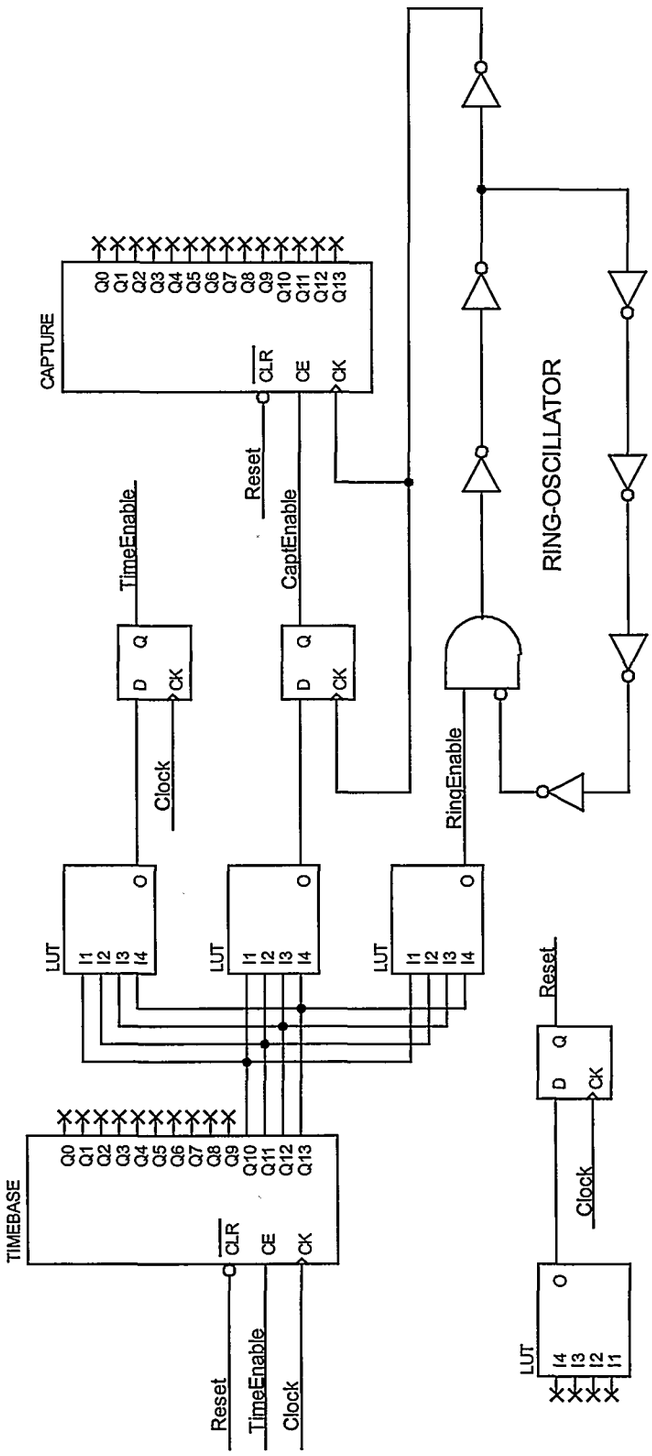



Fig. 77: Diagrama esquemático del sensor

Estas tres habilitaciones se corresponden en el esquemático con las señales `RingEnable`, `CaptEnable` y `TimeEnable`, respectivamente habilitación del oscilador, del contador de captura y del contador de base de tiempos. Estas señales provienen de tres tablas de lookup de 4 entradas, conectadas a los bits más significativos del contador de base de tiempos. Esto significa que se va a ser capaz de fijar la temporización de estas señales en incrementos correspondientes con 1024 pasos de la base de tiempos. Con cuanto tiempo se corresponderán estos incrementos es algo que dependerá de la frecuencia del reloj del sistema. En los experimentos se ha usado un reloj de relativa baja frecuencia, 3,6864 MHz, lo que se traduce en incrementos de 277 $\mu$s. En cualquier caso, los contenidos de estas tres LUTs son parametrizables, para que el sensor sea fácilmente adaptable a sistemas que tengan diferentes frecuencias de reloj.

La temporización usual de estas tres señales será la que se representa en la Fig. 78. El contador de base de tiempos estará habilitado desde el principio (valor de la cuenta 0). Obviamente, si no fuera así el circuito nunca llegaría a funcionar. A continuación se activará el oscilador, de tal manera que transcurridos unos instantes para que la oscilación comience y se estabilice, se activará el contador de captura. Como ya se ha comentado antes, la mayor precisión se alcanzará cuando este tiempo sea tal que en el caso más frío (frecuencia mayor) el contador que justo al límite del desbordamiento. Una vez tomada la frecuencia de oscilación del anillo, se puede deshabilitar. Ya por último, se deshabilitará el contador de base de tiempos, para evitar que se vuelva a reiniciar, con lo que se repetiría el ciclo, y al mismo tiempo, para evitar autocalentamiento, pues ya no es necesario.

Por último, `TimeEnable` está registrada para mejorar la velocidad a la que puede funcionar el contador de base de tiempos (es de esperar que en un sistema real el reloj vaya a decenas de MHz), pues esta señal tendrá un fanout no despreciable (va a todos los CE de los 14 bits del contador), y al mismo tiempo, la latencia que introduce este flip-flop en este circuito no resulta para nada importante. Por otro lado, la señal `CaptEnable` también esta registrada. Aquí el propósito es otro, y es que los dos contadores funcionan con relojes diferentes. En un caso (base de tiempos) usa el reloj del sistema, y en el otro (captura) emplea la salida del oscilador. Para evitar (o reducir al máximo) posibles problemas de metaestabilidad, la señal `CaptEnable` es resincronizada con el reloj que emplea el contador de captura.

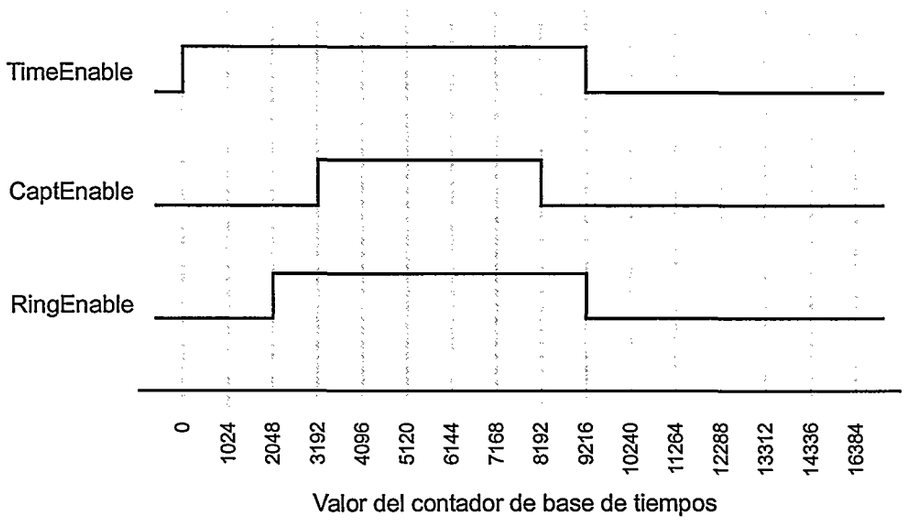



Fig. 78: Temporización típica de las señales `TimeEnable`, `CaptEnable` y `RingEnable`

## 5.1. Manejo del reset

Sorprendentemente, uno de los problemas que se deben resolver al trabajar con reconfiguración en tiempo de ejecución es el manejo del reset. Y es que a diferencia de la metodología convencional de diseño, donde es manejado automáticamente, aquí es el usuario el que debe buscar una solución. En efecto, el funcionamiento 'clásico' de una FPGA es que tras aplicarle tensión comienza a cargarse el bitstream de configuración, y una vez que ha sido cargado, se libera la señal de reset global (GSR), por lo que comienza la actividad normal del circuito. Este comportamiento, en el cual la señal de reset global se activa durante la configuración de la FPGA, es incompatible con la reconfiguración parcial en tiempo de ejecución. Cuando se produce la reconfiguración de una parte del circuito no se puede resetear toda la FPGA, pues esto impediría que el resto del circuito, que debe continuar su operación normal, siguiese funcionando correctamente. Mientras que esta limitación no presenta mayor problema cuando estemos introduciendo circuitos combinacionales, o reconfigurando la parte combinacional de un circuito, como por ejemplo las LUTs que contienen los coeficientes de un filtro adaptativo, el problema viene cuando el nuevo circuito que se carga en la FPGA hace uso de los registros y necesita que esto estén inicializados a un valor concreto.

La solución que se ha utilizado para resolver este problema es generar un reset local que vaya junto con el circuito que se va a introducir en la FPGA. Esta señal se generará en una LUT, que al principio estará configurada de tal manera que el reset esté activo. De esta manera, cuando se introduzca el circuito en la FPGA se inicializará en el estado correcto. Para que el circuito comience su funcionamiento normal, se debe desactivar la señal de reset. Esto se consigue reconfigurando la LUT que genera el reset. A partir de este momento el circuito ya podrá comenzar su funcionamiento normal.

Los pasos serán entonces los siguientes:

1. En una primera configuración se carga todo el circuito, con su señal de reset local activada. El circuito no llega a funcionar, pues el reset lo mantiene continuamente en su estado inicial.
2. A continuación se reconfigura la LUT que genera el reset, para desactivarlo. Esta será una operación muy rápida, pues sólo habrá que modificar 16 *frames* de la FPGA. A partir de este momento comienza a operar el circuito
3. Una vez que el circuito haya finalizado la tarea que debía realizar, con una nueva reconfiguración puede ser eliminado de la FPGA. Alternativamente, se puede reconfigurar la LUT que genera el reset para volver a activarlo, de tal manera que se vuelva al estado inicial; al estar inhibida la actividad se puede eliminar prácticamente en su totalidad el consumo de potencia del circuito.

Estos pasos se resumen en la Fig. 79. Puesto que la reconfiguración de la LUT que genera el reset es asíncrona, se ha registrado su salida con objeto de evitar problemas de metaestabilidad, tal y como se hizo con la señal `CaptEnable`.

## 5.2. Lectura del valor de temperatura

Al finalizar todo este proceso, el contador de captura tendrá un valor proporcional a la frecuencia de oscilación del anillo, y por tanto, a la temperatura. Como se puede ver en la Fig.2, las salidas de este contador no van a ningún sitio. Entonces, ¿cómo se lee este valor?. La respuesta es fácil: usando readback. A través del puerto de configuración es posible leer el estado de todos los flip-flops de la FPGA, y este es el método que se utiliza para ver en que punto se ha quedado este contador. Así no es necesario utilizar ninguna pata de E/S; toda la operación se hace a través del puerto de configuración.

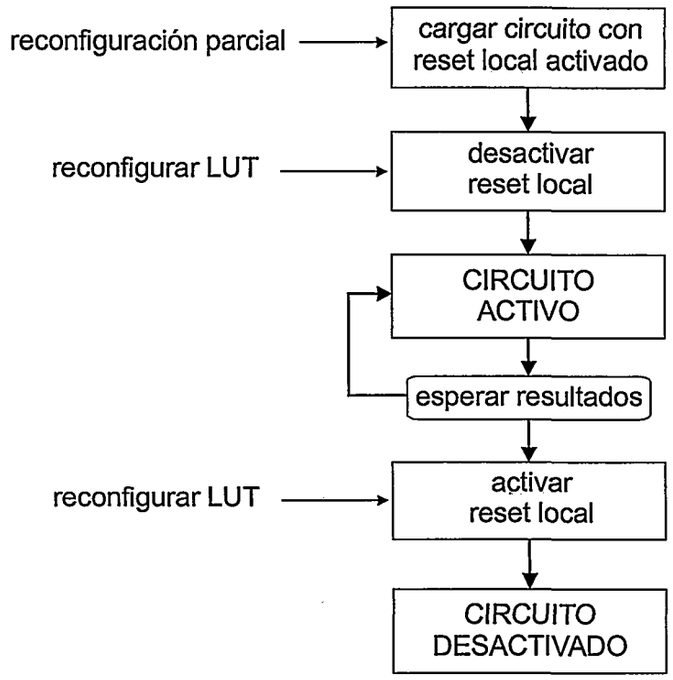



Fig. 79: Esquema de la operación del reset propuesta para arquitecturas RTR

## 5.3. Layout

En la Fig. 80 se puede observar el layout del sensor; para mayor claridad se han sombreado los *slices* utilizados. Como se puede comprobar su tamaño es muy pequeño, de 8 CLBs de alto por 2 de ancho. Este tamaño representa desde el 4,2% del dispositivo más pequeño de la familia, la XCV50, hasta el 0,26% del más grande, la XCV1000. O sea, que el sensor es lo suficientemente pequeño como para que sea factible utilizarlo en aplicaciones de mapa térmico del dispositivo.

A continuación, en la Fig. 81 se representa la funcionalidad de cada *slice* utilizado por el sensor. Se puede ver como se ha dejado un espacio mínimo de dos *slices* entre cada uno de los inversores del oscilador en anillo, para tratar de aumentar el rutado y disminuir así la frecuencia de oscilación.

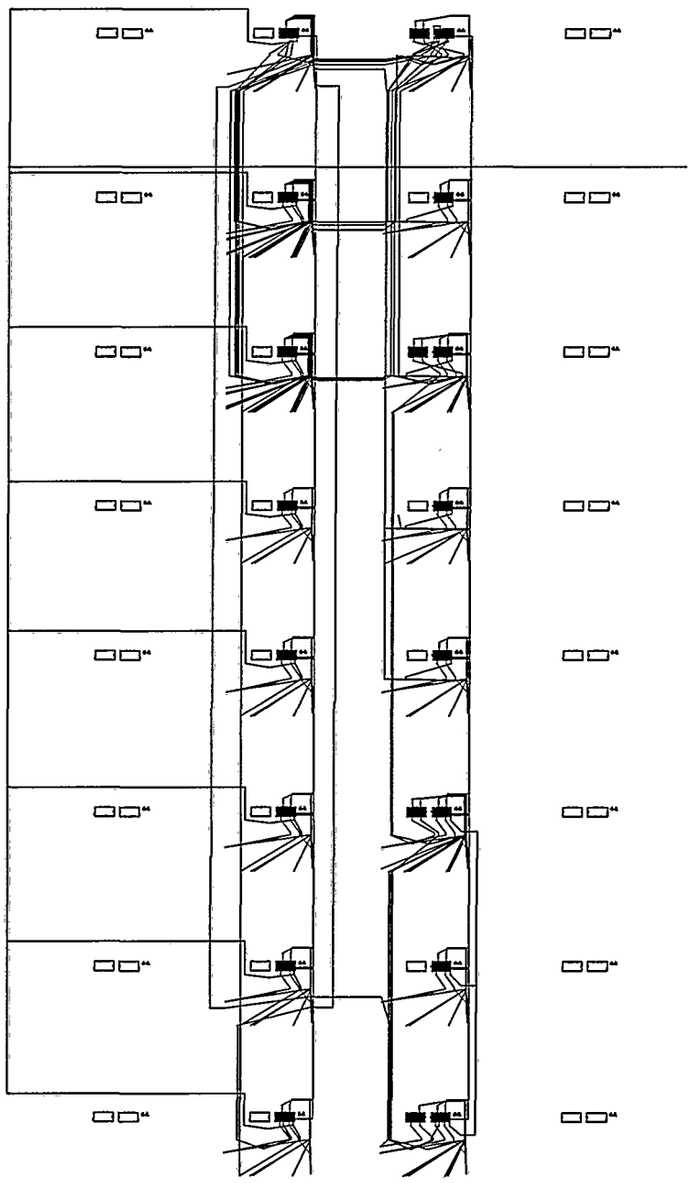


Fig. 80: Layout del sensor compatible con RTR

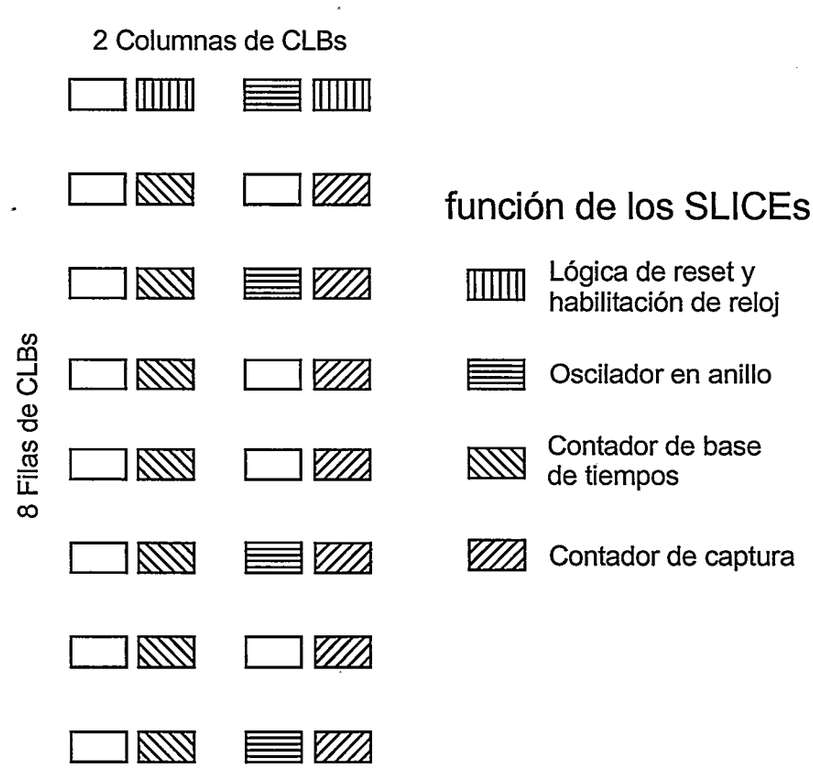



Fig. 81: Función de los *slices* empleados por el sensor

## 5.4. Detalles de codificación

El sensor ha sido codificado como un `RTPCore`, siguiendo el API `CoreTemplate`. Es un core primitivo, es decir, no usa otros cores, sino que accede directamente a los detalles de bajo nivel de la FPGA. La ventaja que tienen estos cores es que permiten hacer un control muy estricto de los recursos que se utilizan; el inconveniente es lo complejo que resultan de programar. En el apéndice B se puede encontrar el código del sensor.

Los únicos parámetros que se le deberán pasar al sensor son las temporizaciones de las tres señales de habilitación `TimeEnable`, `RingEnable` y `CountEnable`, en forma de los contenidos de las LUTs que los generan. Estos parámetros se pasan en la llamada al método `implement`. A modo de ejemplo, este el código utilizado en los experimentos para instanciar los sensores:

```
TemperatureSensor sensor;
int[] TimeEnable  = {1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0};
int[] RingEnable  = {0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0};
int[] CaptEnable  = {0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0};
// instantiate the sensor
sensor = new TemperatureSensor("Sensor", clk);
// place the sensor
Offset sensorOffset = sensor.getRelativeOffset();
sensorOffset.setHorOffset(Gran.CLB, 4);
sensorOffset.setVerOffset(Gran.CLB, 4);
// implement the sensor
sensor.implement(TimeEnable, RingEnable, CaptEnable);
```

Como se puede ver, las temporización de las señales `TimeEnable`, `RingEnable` y `CountEnable` se define en forma de matrices de enteros. Para comprender mejor como se especifican, lo mejor es volver a la Fig. 78, donde se ve que el tiempo se divide en 16 fracciones. Cada una de esta divisiones dura 1024 periodos del reloj del sistema. Si se quiere que la señal esté a 1 en la división i, entonces el elemento i de su matriz de enteros debe ser también 1. Y si no, debe ser 0. En el ejemplo, la señal `CaptEnable` está a 0 durante los 2048 primeros ciclos, luego se activa durante 1024 ciclos, y luego se desactiva definitivamente. Obviamente, para que el sensor funcione correctamente la temporización debe seguir lo especificado en el punto 5, y en particular, la forma de la señales debe ser similar a la de la Fig. 78.

Por último, el tamaño del core sensor es siempre el mismo, 8 por 2 CLBs, independientemente de los parámetros.

# 6. Experimentos realizados

Para caracterizarlo, el sensor de temperatura se implementó en dos FPGAs de la familia Virtex: la XCV50PQ240-4C, y la XCV800HQ240-4C. Como plataforma de desarrollo se utilizó una tarjeta Xilinx AFX [Xil99d], cuyo aspecto se puede verse en el apéndice B. Esta tarjeta es muy sencilla, básicamente está compuesta por un zócalo para poner la FPGA, conectores para alimentarla y programarla, otro zócalo por si se quiere

usar una EPROM de programación, un área de grapinado, y una serie de pines que facilitan el acceso a todas las patas de la FPGA. El concepto de diseño de esta placa es completamente distinto al de otras tarjetas de prototipado para Virtex mucho más conocidas, como la XESS XCV [Xes01]. Estas placas incluyen una gran cantidad de periféricos (memorias, Ethernet, conversores A/D y D/A, interfaces de vídeo...), que facilitan mucho el diseño de sistemas. Aunque la falta de estos dispositivos pueda parecer un grave inconveniente de la tarjeta AFX, lo cierto es que para los experimentos en los que se iba a emplear (medidas de consumo y de temperatura) es todo lo contrario. En una tarjeta con periféricos es imposible (o muy difícil) medir el consumo, porque normalmente no disponen de planos de alimentación separados para la FPGA y los periféricos. Además, el calentamiento de los otros circuitos que pueda tener la placa puede interferir en la medidas de temperatura.

Esta tarjeta no tiene soporte para JBits, que fue necesario desarrollarlo. La elección entre las dos interfaces de la FPGA que permiten reconfiguración parcial fue inmediata, la paralela *SelectMap* es más rápida, y sobre todo, mucho más sencilla de manejar que la serie JTAG. Para conectar la FPGA al PC corriendo JBits se escogió usar el puerto paralelo, por su sencillez, aunque el precio a pagar fue una pobre velocidad de configuración (centenas de $KBs^{-1}$). Más detalles acerca de este desarrollo pueden encontrarse en el apéndice B.

Para calibrar los sensores se utilizó un montaje similar al descrito en el capítulo anterior, utilizando un horno de temperatura controlada y midiendo la temperatura del encapsulado con un termopar. Una vez más, en el apéndice B puede observarse el montaje que se habilitó.

En el caso de la FPGA más pequeña, la XCV50, los experimentos se realizaron con un único sensor localizado en una esquina, mientras que en la XCV800 se colocaron 5 sensores, uno en el centro y cuatro en las esquinas. A continuación se muestran los resultados experimentales; para cada uno de los casos se muestra la respuesta tanto en temperatura como frente a variaciones en la tensión de alimentación interna.

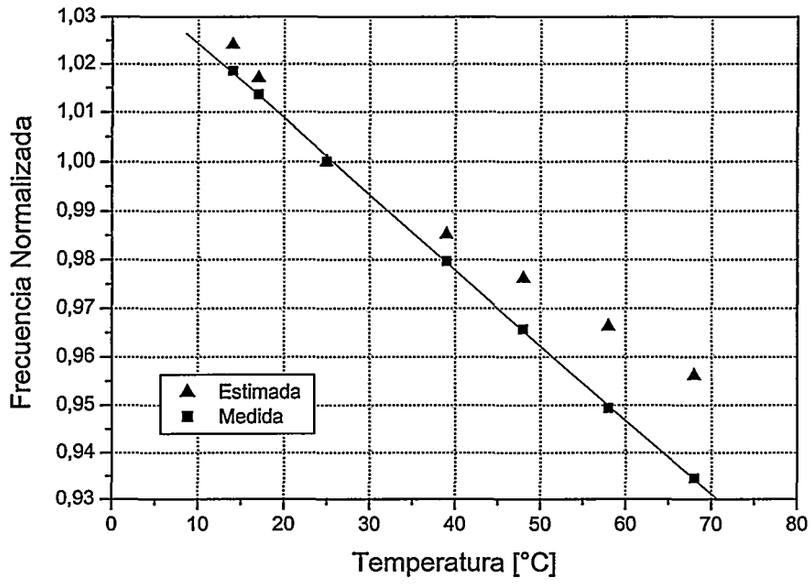



Fig. 82: Frecuencia normalizada (real y estimada) vs. temperatura del sensor JBits en XCV50

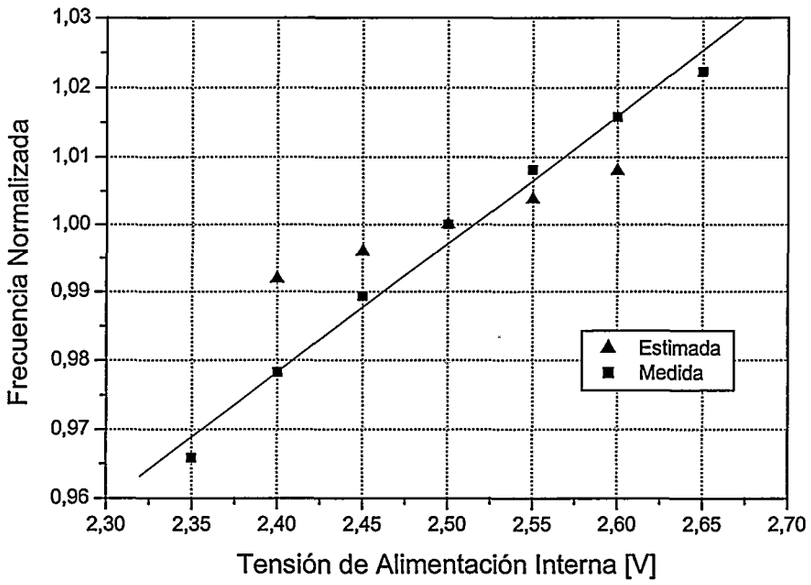



Fig. 83: Frecuencia normalizada (real y estimada) vs. Vcc del sensor JBits en XCV50

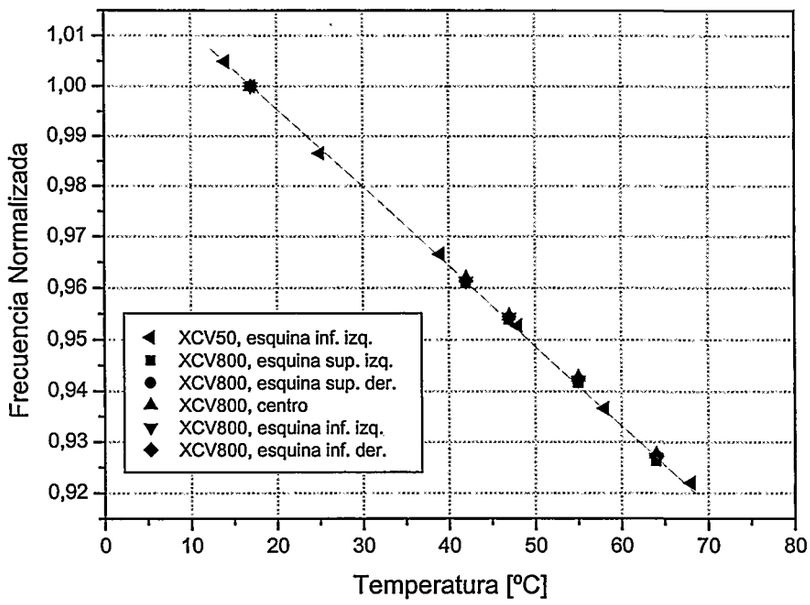



Fig. 84: Frecuencia normalizada vs. temperatura del sensor JBits en XCV50 y XCV800

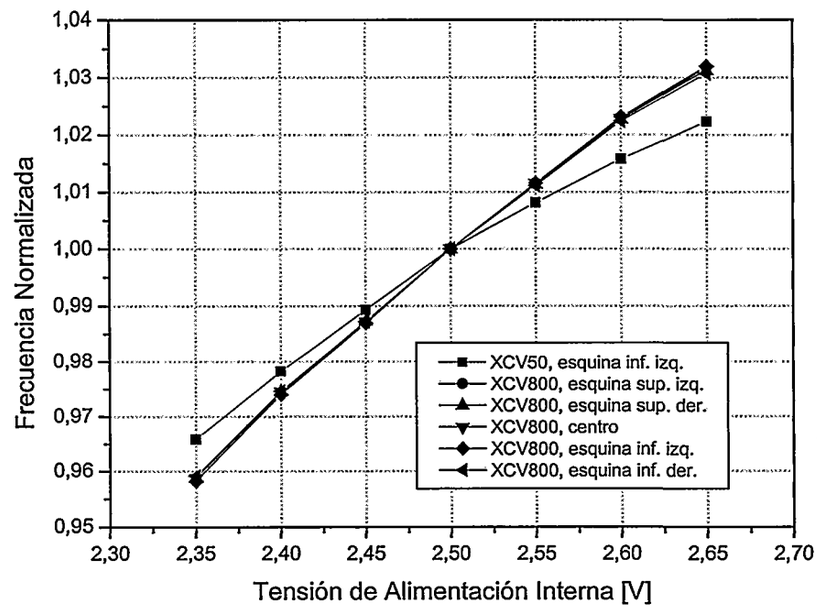



Fig. 85: Frecuencia normalizada vs. Vcc del sensor JBits en XCV50 y XCV800

En todas las gráficas se ha utilizado la frecuencia normalizada; una de las conclusiones de los experimentos del capítulo anterior es que no tiene mucho sentido representar las frecuencias absolutas, pues cada sensor oscila a una frecuencia distinta, incluso aunque tengan el mismo layout. Por esta razón, si se representan los valores absolutos no resulta evidente ver si los todos los sensores se comportan igual; un buen ejemplo de esto es la Fig. 45. Ya que se sabe que las frecuencias absolutas van a ser diferentes, interesará al menos saber si el coeficiente de variación es igual para todos los sensores, y por eso se representa la frecuencia normalizada. De cualquier manera, en la siguiente tabla se indican las frecuencias absolutas

| **XCV50 esq. inf. izq.** | **XCV800 esq. sup. izq.** | **XCV800 esq. sup. der.** | **XCV800 centro** | **XCV800 esq. inf. izq.** | **XCV800 esq. inf. izq.** |
|---|---|---|---|---|---|
| 46,148 MHz | 40,576 MHz | 40,946 MHz | 40,936 MHz | 41,116 MHz | 42,120 MHz |
| @ 2,5V 17°C | | | | | |

Tabla 11: Frecuencias absolutas de oscilación de los sensores JBits

En la Fig. 82 y en la Fig. 83 se representa el comportamiento real del sensor en una XCV50 junto con los valores estimados, empleando la opción de prorrateo tanto en temperatura como en tensión de la herramienta de análisis estático de tiempos (Foundation 3.1). Esta herramienta predice una frecuencia de oscilación de 33,887 MHz a 2,5V y 17 °C, correspondiente con un retardo de 4,725 ns para la lógica, y 10,030 ns para el rutado. Este dato, junto a los resultados mostrados en las figuras antes mencionadas, no hace más que corroborar la conclusión obtenida en el capítulo anterior: las herramientas de análisis estático de tiempos son de poca utilidad a la hora de trabajar con osciladores en anillo.

Los datos que se muestran muy interesantes son los de la Fig. 84 y la Fig. 85. Estos resultados parecen indicar que los sensores con idéntico layout tienen el mismo coeficiente de variación frente a la temperatura, incluso aunque se cambie de dispositivo. El comportamiento frente a las alteraciones en la tensión de alimentación es igual dentro de una misma FPGA, pero varía si se cambia de chip. Estas conclusiones son las mismas que se obtuvieron al comparar entre las FPGAs XC4005E y XC4005, y podrían indicar que es un comportamiento que se puede generalizar a todas las FPGAs.

# 7. Modificaciones al diseño: rutado largo

En varios experimentos del capítulo anterior se puede ver como las variaciones en el rutado afectan a la respuesta de los osciladores en anillo, tanto para la temperatura como para las variaciones de Vcc. Esto es debido a que en las FPGAs el rutado se realiza principalmente mediante llaves CMOS, que tienen unas características distintas a las de los búferes de salida de los elementos programables. Así, para un rutado corto, la respuesta global del sensor se parecerá más a la de los búferes de salida, y en el caso de un rutado largo, a la de las llaves CMOS.

Puesto que existen estas variaciones en la respuesta, lo que parece entonces más adecuado es hacer distintas pruebas con varios sensores, hasta dar con el que mejores características tenga: mayor sensibilidad con respecto a la temperatura y menor con respecto a Vcc, menor tamaño, etc... Pero, por lo visto en los experimentos que se han mostrado hasta ahora, la respuesta de todos los sensores es bastante parecida. Y en la práctica, lo que normalmente ocurrirá es que el primero que se pruebe será lo suficientemente bueno, y no será muy interesante gastar tiempo en buscar otras alternativas.

Sin embargo, si que puede existir un caso que justifique investigar la respuesta de varios sensores. Tal y como se indica en [Que91], si se consigue tener al menos dos sensores cuya sensibilidad a Vcc sea suficientemente diferente, se puede llegar a conocer simultáneamente la tensión y la temperatura de operación del integrado. El formalismo matemático que se utiliza es muy sencillo: se modela la respuesta de los dos sensores como una función dependiente de T y Vcc, y una vez medida la frecuencia de ambos osciladores, se resuelve el sistema de ecuaciones para obtener el resultado. Sin duda esta posibilidad es muy interesante, pues significaría tener resuelto el mayor problema que presentan los osciladores en anillo: la incertidumbre que provocan las variaciones en Vcc.

En este punto lo que se ha tratado de evaluar es si es posible modificar el sensor que se ha presentado en este capítulo para conseguir una respuesta frente a Vcc diferente, pero sin variar sus características principales: tamaño y modo de operación. La estrategia que se ha seguido para aumentar el rutado es muy sencilla: simplemente se han cambiado de lugar los inversores, de tal manera que la posición en la que están ahora no

sea tan favorable para el rutado. En la Fig. 86 se puede observar la disposición original y la modificada. Con este sencillo cambio se consigue aumentar en un 24,8% el retardo del rutado, y un incremento global del retardo de 16,8%, pasando a ser ahora 12,520 ns y 17,245 ns respectivamente, según los datos proporcionados por la herramienta de análisis estático de tiempos a 2,5V y 17°C.

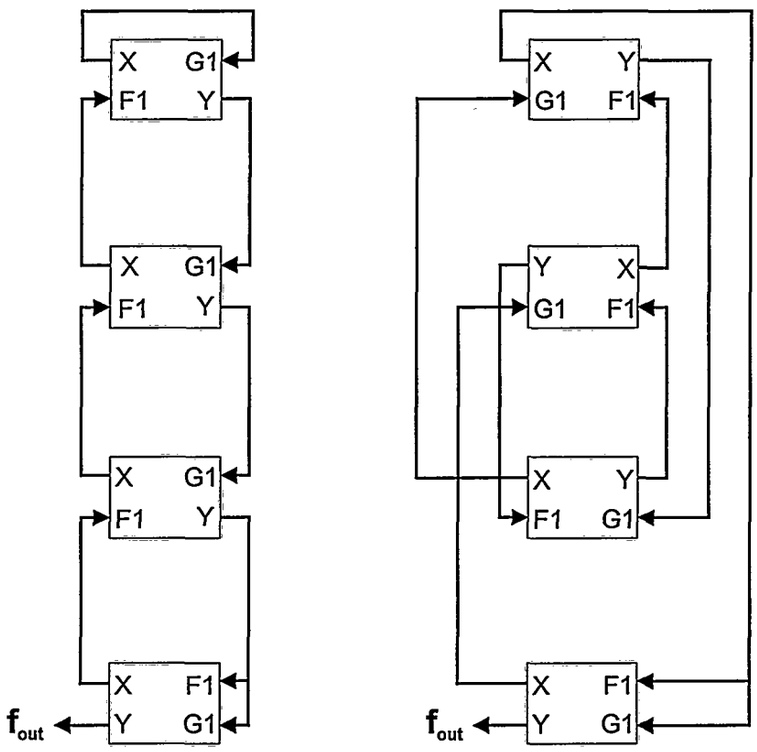



Fig. 86: Modificaciones en el emplazamiento del oscilador en anillo para aumentar el rutado

Aunque será en el punto 9 donde se presentarán con detalle los experimentos realizados, en la Fig. 87 y en la Fig. 88 se puede ver como es la respuesta del sensor normal frente al modificado. Como se puede ver, aunque la modificación ha sido muy sencilla, si que se ha conseguido variar apreciablemente la respuesta del sensor frente a Vcc. Además, esta variación ha sido provechosa, pues se reduce apreciablemente la sensibilidad. Por otro lado, la respuesta frente a la temperatura permanece prácticamente igual.

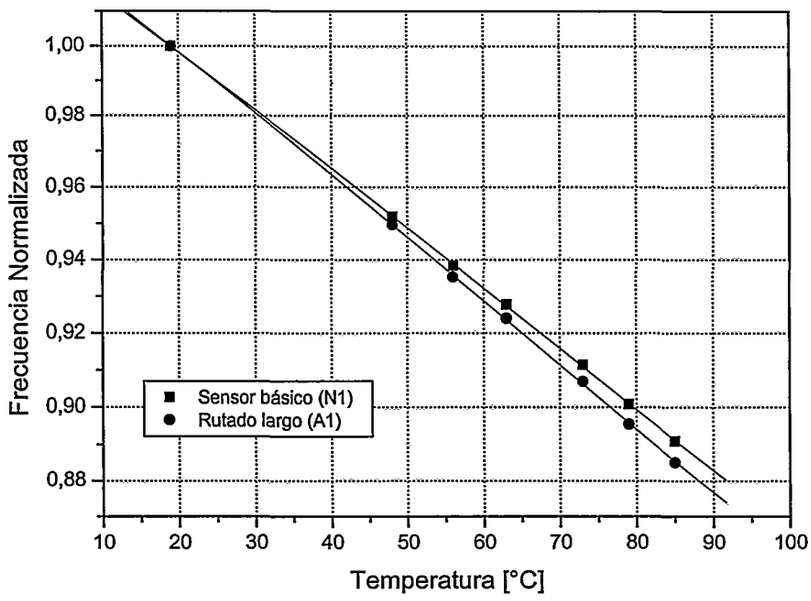



Fig. 87: Comparación en la respuesta vs. temperatura de los sensores con dos tipos de rutado

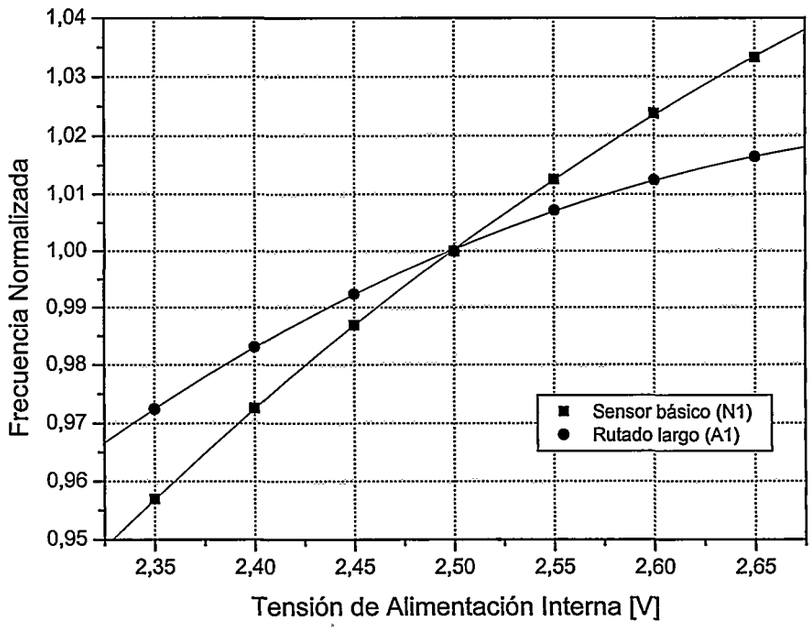



Fig. 88: Comparación en la respuesta vs. Vcc de los sensores con dos tipos de rutado

La respuesta de ambos sensores se puede modelar según la ecuación:

$$f_n = (a + bV + cV^2)(f + gT)$$

Donde $f_n$ es la frecuencia normalizada, V es la tensión interna [V] y T es la temperatura [°C]. Haciendo un ajuste de las curvas de las figuras anteriores, se obtiene que los coeficientes serían en cada uno de los casos:

| Sensor | a | b | c | f | g |
|---|---|---|---|---|---|
| N1 | -1,04999 | 1,38524 | -0,22605 | 1,03115 | -0,00165 |
| A1 | -0,98182 | 1,4389 | -0,25842 | 1,03304 | -0,00174 |

Tabla 12: Coeficientes de la curva que modela la frecuencia normalizada de los sensores N1 y A1

Estos coeficientes se han obtenido de la medida a tensión constante (2,5 V) en 7 temperaturas: 19, 48, 56, 63, 73, 79 y 85 °C, y de los valores a temperatura constante (19 °C) con 7 tensiones: 2,35, 2,40, 2,45, 2,50, 2,55, 2,60 y 2,65 V.

Resolviendo el sistema de ecuaciones se llega a una ecuación de 4º grado. No es este el camino más adecuado, pues la solución simbólica de estas ecuaciones es muy enmarañada; la opción que se ha tomado es resolverlo con Mathcad. De las cuatro soluciones, tres se eliminan inmediatamente: bien porque son complejas o porque no tienen sentido físico. Como el rango de temperaturas y tensiones válidas es relativamente estrecho para el comportamiento suave de las funciones, no hay posibilidad de que haya dos soluciones válidas

Una vez creado el modelo, el último paso (y el principal) es comprobar si sus resultados se ajustan a la realidad. Para ello se han utilizado las medidas tomadas en los puntos anteriormente mencionados; aunque en el fondo es un tanto engañoso, porque se están utilizando los valores que se han utilizado para crear el modelo. Por eso, también se han utilizado medidas en las 7 diferentes tensiones a otras dos temperaturas: 73 y 85 °C. Los resultados se muestran desde la Tabla 13 a la Tabla 16, y son muy buenos: se consiguen errores de menos de ±1 °C en la temperatura. Para la tensión, los resultados son todavía mejores: menos de ±5 mV de error.

| @ 2,50 V | 19 °C | 48 °C | 56 °C | 63 °C | 73 °C | 79 °C | 85 °C |
|---|---|---|---|---|---|---|---|
| Tensión [V] | 2,501 | 2,498 | 2,499 | 2,500 | 2,499 | 2,501 | 2,501 |
| Temperatura [°C] | 19,29 | 47,92 | 56,28 | 62,78 | 72,58 | 79,28 | 85,29 |

Tabla 13: Valores pronosticados por el modelo para distintas temperaturas y tensión de 2,50 V

| @ 19 °C | 2,35 V | 2,40 V | 2,45 V | 2,50 V | 2,55 V | 2,60 V | 2,65 V |
|---|---|---|---|---|---|---|---|
| Tensión [V] | 2,352 | 2,401 | 2,451 | 2,501 | 2,551 | 2,603 | 2,650 |
| Temperatura [°C] | 19,25 | 19,09 | 18,99 | 19,29 | 18,99 | 19,08 | 19,04 |

Tabla 14: Valores pronosticados por el modelo para distintas tensiones y temperatura de 19 °C

| @ 73 °C | 2,35 V | 2,40 V | 2,45 V | 2,50 V | 2,55 V | 2,60 V | 2,65 V |
|---|---|---|---|---|---|---|---|
| Tensión [V] | 2,352 | 2,402 | 2,450 | 2,499 | 2,550 | 2,601 | 2,652 |
| Temperatura [°C] | 72,55 | 72,40 | 72,23 | 72,58 | 72,39 | 72,49 | 72,59 |

Tabla 15: Valores pronosticados por el modelo para distintas tensiones y temperatura de 73 °C

| @ 85 °C | 2,35 V | 2,40 V | 2,45 V | 2,50 V | 2,55 V | 2,60 V | 2,65 V |
|---|---|---|---|---|---|---|---|
| Tensión [V] | 2,353 | 2,401 | 2,449 | 2,501 | 2,554 | 2,603 | 2,653 |
| Temperatura [°C] | 85,09 | 85,16 | 84,89 | 85,29 | 85,40 | 85,48 | 85,49 |

Tabla 16: Valores pronosticados por el modelo para distintas tensiones y temperatura de 85 °C

# 8. Compatibilidad con el flujo convencional de diseño

Si bien JBits es una metodología muy potente, porque permite crear diseños parametrizables en tiempo de ejecución, su punto débil es que es un flujo de diseño completamente nuevo, por lo que se desperdician muchas de las facilidades que ofrecen las herramientas de diseño convencionales. Lo óptimo sería poder utilizar todas las posibilidades que ofrece JBits, en especial la reconfiguración en tiempo de ejecución, sin tener que renunciar a la potencia de las herramientas clásicas de síntesis.

Sin embargo, la coexistencia de dos metodologías de diseño nunca es una tarea sencilla, y este caso no es una excepción. En el contexto de esta tesis, lo ideal sería poder añadir dinámicamente sensores de temperatura a un diseño realizado con las herramientas convencionales, todo ello en tiempo de ejecución. La manera que tiene JBits de facilitar esta posibilidad es con el uso de *anticores*. Básicamente, los *anticores* permiten crear 'huecos' en el diseño que luego se pueden rellenar con lógica descrita en JBits. Así, para utilizar un core JBits en un diseño convencional lo que se hace es generar su correspondiente *anticore*. Este *anticore* conseguirá que tras el emplazado y rutado clásico quede un área libre en la configuración de la FPGA, en la que sólo habrá el rutado que haga falta para conectar el core con el resto del diseño. Más tarde, en JBits se lee este bitstream y se rellena el 'hueco' con el core original. De esta manera se consigue integrarlo en un diseño hecho con las herramientas convencionales de síntesis. Para más detalles, en el apéndice A se hace una descripción completa de su funcionamiento.

## 8.1. Evaluación de la alternativa del uso de *anticores*

En un principio debería ser sencillo utilizar la técnica de los *anticores* con los sensores de temperatura que se han descrito en este capítulo. Además, como no tienen conexiones externas, no hay que preocuparse de cómo se hará el rutado entre el sensor y el resto del diseño, porque directamente es que no existe.

Para evaluar si esta estrategia funciona correctamente, se ha creado un nuevo core con el mismo tamaño que el sensor, 8 por 2 CLBs; esto es necesario porque no se puede generar un *anticore* a partir del sensor original. Por una limitación de JBits, los cores que acceden directamente a la configuración de la FPGA (con `jbits.set`) no sirven para crear *anticores*; sólo es posible con los que están descritos a base de instancias de las *ULPrimitives* (que son elementos básicos tales como LUTs, flip-flops, multiplexores

internos del *slice*, memorias...). Y así, este nuevo core está formado por tablas de lookup y flip-flops interconectados entre si, sin una funcionalidad determinada. Los *anticores* son independientes de los contenidos del core, sólo dependen de su tamaño y de sus conexiones externas. Por eso no era necesario hacer el esfuerzo de rescribir completamente el sensor en términos de *ULPrimitives*; sólo con crear un core con su mismo tamaño y que utilizase estas primitivas era suficiente.

Una vez creado este core, se siguió todo el proceso estándar:

- El core se instanció en un nuevo diseño
- Se creó el correspondiente *anticore*
- A continuación, se utilizó el *anticore* dentro de un diseño convencional
- Una vez obtenido el bitstream con las herramientas convencionales, se eliminó el *anticore* utilizando el código generado automáticamente por JBits
- Ya otra vez de vuelta en JBits, se instanció un sensor de temperatura en el hueco dejado por el *anticore*
- Y por último, se escribió el nuevo bitstream, que incluía el sensor JBits integrado en el diseño convencional.

Una vez descargado el bitstream en la FPGA, se comprobó que el diseño original (el realizado con las herramientas convencionales) no estaba funcionando correctamente. Rastreando el origen del problema, se llegó a la conclusión de que la causa era que al rutar el sensor dentro de JBits se estaba estropeando el conexionado de nodos del diseño original. Aunque JBits en teoría tiene un mecanismo para descubrir los caminos en un bitstream ya existente (`ResourceFactory`), todo parece indicar que no estaba funcionando correctamente.

Por lo tanto, la conclusión es que el método de los *anticores*, que en un principio parecía el más adecuado para utilizar los sensores que se han descrito en este capítulo en diseños convencionales, no es válido por limitaciones del propio JBits. Por eso es necesario buscar una alternativa.

## 8.2. Modificaciones al diseño: transformación en macro física

Una alternativa compatible con el flujo de diseño clásico es describir el sensor de temperatura como una macro, creando un circuito completamente equivalente al descrito con JBits, pero utilizando las herramientas convencionales. La única característica que se perdería es la posibilidad de añadir o eliminar el sensor en tiempo de ejecución, pero el resto de su funcionamiento permanecería igual: el sensor se activaría usando reconfiguración parcial, y sus resultados se leerían mediante readback. Sin duda esta limitación elimina uno de los aspectos más novedosos del sensor: uno de sus puntos más interesantes es la capacidad de insertarlo o retirarlo dinámicamente de la FPGA. Pero la posibilidad de tener un sensor totalmente autocontenido, que se maneje a través del puerto de configuración, sin usar pines adicionales, y que además haya sido creado con las herramientas convencionales, parece suficientemente atractiva como para que sea explorada en esta tesis.

El requerimiento de describir el sensor como una macro viene de la necesidad de tener el emplazado controlado. Primero, para tratar de mantener el rutado lo más uniforme posible en todos los sensores. Y segundo, para poder conocer fácilmente la localización de las tablas de lookup que controlan su funcionamiento, sin tener que bucear en los informes de emplazado. Obviamente, se podría crear el sensor sin utilizar una macro, pero el esfuerzo que se ahorraría no compensaría los problemas que surgirían por no tener el emplazado fijo.

A la hora de crear una macro con las herramientas de Xilinx, hay dos alternativas: usar emplazamiento relativo, o macros físicas. La primera se basa describir el diseño en base a primitivas de bajo nivel (LUTs, flip-flops, cadenas de acarreo...) y fijar la posición relativa de unas respecto a otras. Para ello se puede emplear la captura de esquemáticos o la síntesis, utilizando un HDL estructural. La otra opción se basa en describir a bajo nivel el circuito, programando manualmente los recursos del chip con el FPGA editor.

La ventaja de las macros con emplazamiento relativo (RPMs) es que son más sencillas de crear que las macros físicas, para las que hace falta un duro trabajo frente al FPGA editor. Las RPMs son especialmente útiles para diseños muy regulares, pues se pueden crear muy bien desde el HDL utilizando instanciación iterativa, y al hacerse un control estricto del emplazado, se consiguen muy buenos rendimientos. La ventaja de las macros físicas es que se tiene un control absoluto del diseño: no sólo del emplazado,

sino también del rutado. A diferencia de las RPM, las macros físicas no se vuelven a rutar con cada implementación, por lo que sus características se mantienen invariables. Aunque en la realidad esto no es un problema, porque el emplazado prácticamente define el rutado, por lo que las variaciones, si las hay, serán pequeñas. Sólo en el caso de que el resto del diseño tenga un rutado muy denso, puede ocurrir que llegue a 'invadir' el area de la macro, ocasionando que aumenten los retardos en el rutado de la propia macro. Pero como ya se ha indicado, en las FPGAs actuales, ricas en recursos de rutado, este problema no debe ser considerado ni mucho menos como grave. En el contexto de esta tesis, si que las variaciones en el rutado hay que tenerlas en cuenta, porque pueden alterar la respuesta del sensor. Pero como se vio en el punto anterior, afectan más a la sensibilidad frente a Vcc que a la respuesta en temperatura.

En cualquier caso, se ha optado por evaluar la conversión del sensor en macro física. La razón es que para un circuito tan sencillo realmente no hay mucha diferencia de dificultad entre esta opción y usar una RPM; realmente es sólo en el caso de diseños muy grandes y regulares donde las RPM resultan mucho más sencillas de implementar. Y además, con una macro física se elimina toda incertidumbre en el rutado (aunque, como ya se ha indicado, este punto tampoco es muy crítico). El proceso que se ha seguido para crear la macro es totalmente directo: se ha clonado en el editor de FPGAs la configuración descrita en el código JBits. Puesto que la macro es equivalente al sensor original, se puede usar el mismo código JBits para manejarlo y leer sus medidas.

A parte de la macro correspondiente con el sensor original, se ha creado otra a partir de la versión modificada para aumentar el rutado que se ha descrito en el punto anterior. Sus layouts pueden observarse respectivamente en la Fig. 89. Usar estas macros es absolutamente trivial; este sería el código VHDL que se emplearía para instanciarlas:

```
architecture some of example is
...
     component TempSensor port (clk: in std_logic);
     end component;
...
begin
...
     s00 : TempSensor port map (clk => buf_clk_sens);
```

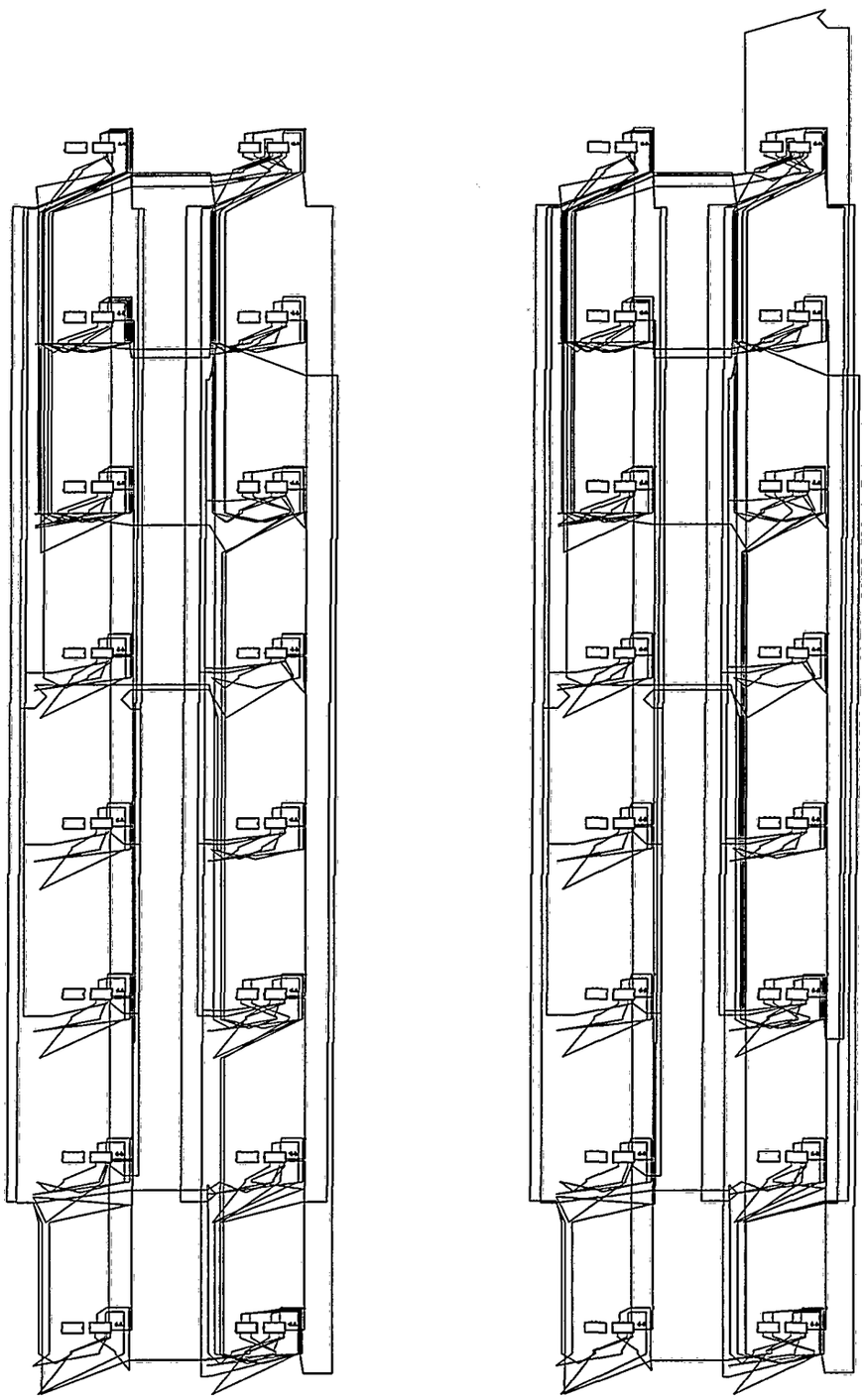


Fig. 89: Layout de la macro sensor (Izqa.) y de su modificación con rutado largo (Drcha.)

Lo único que hay que tener en cuenta es que es conveniente añadir una directiva de emplazamiento absoluto (LOC) para fijar la posición de la macro dentro de la FPGA. Todas las macros tienen un componente (*slice*) de referencia, que es el que se usa junto la directiva LOC para establecer su posición. En este caso, la referencia está situada en la esquina inferior izquierda de la macro; de esta manera, si se quisiera colocar en la esquina inferior izquierda de una XCV800 se utilizaría el siguiente atributo en VHDL:

```
attribute LOC : string;
attribute LOC of s00 : label is "CLB_R56C1.S0";
```

O bien, la siguiente directiva en el archivo UCF:

```
INST s00 LOC = "CLB_R56C1.S0";
```

# 9. Modificaciones al diseño: experimentos realizados

En este punto se ha tratado de evaluar como afectan las modificaciones descritas en los dos puntos anteriores a la respuesta del sensor. Para ello se han caracterizado un total de 16 circuitos, correspondientes con cuatro instancias de los cuatro tipos de sensores:

- Normal JBits, el que sirve de base a este capítulo (N)
- JBits con rutado largo, descrito en el punto 7 (A)
- Macro con rutado normal, equivalente al normal JBits (M)
- Macro con rutado largo, similar al JBits con rutado largo (MA)

Estos 16 sensores se han colocado equiespaciados sobre una XCV800HQ240-4C alternando distintos tipos de sensores. Esta disposición puede verse mejor en la Fig. 90. A continuación, desde la Fig. 91 hasta la Fig. 98 se muestran los resultados experimentales, agrupados por sensores del mismo tipo. Por último, en la Fig. 99 y en la Fig. 100 se resumen los resultados, graficándose la respuesta de los 16 sensores tanto frente a temperatura como a variaciones en la tensión interna de alimentación.

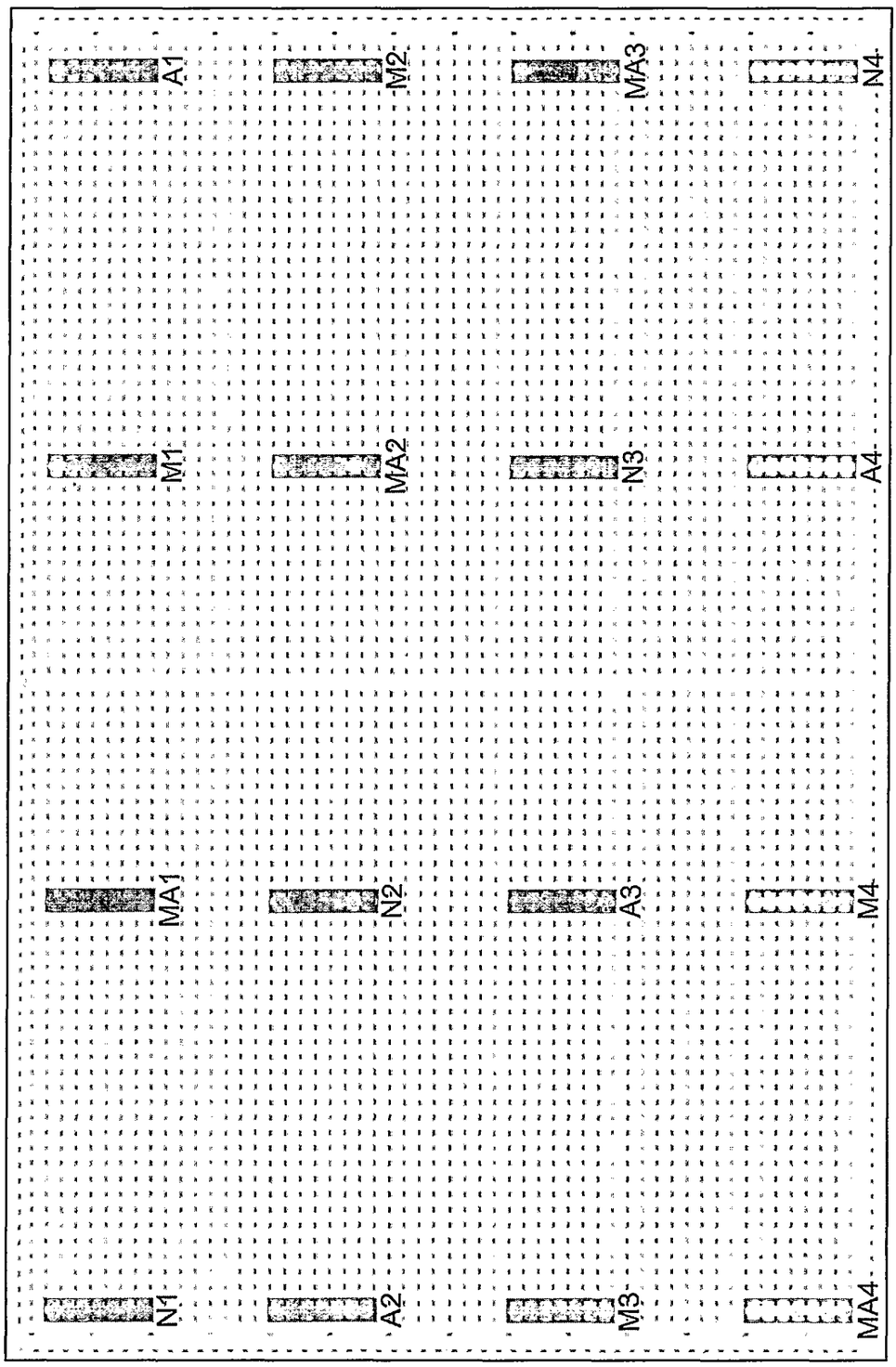



Fig. 90: Disposición de los sensores en la FPGA

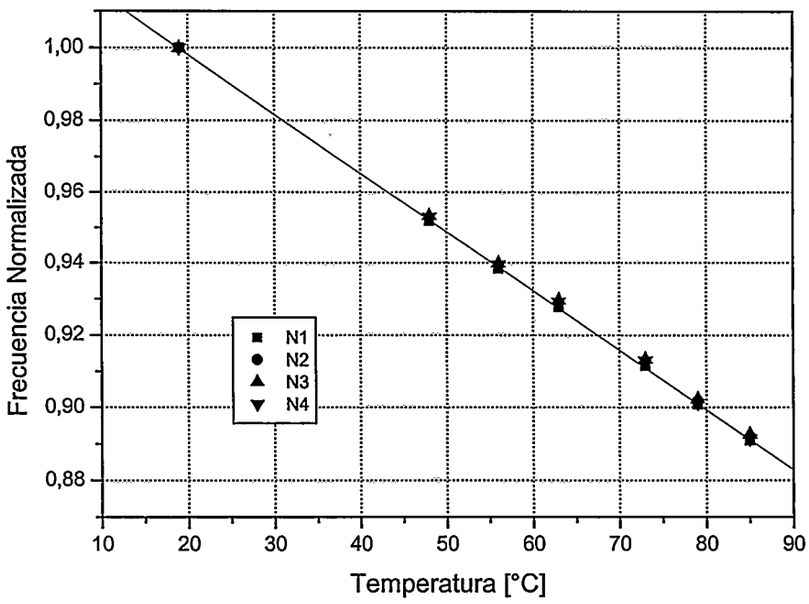



Fig. 91: Frecuencia de salida vs. temperatura en los sensores N1 a N4 (básicos)

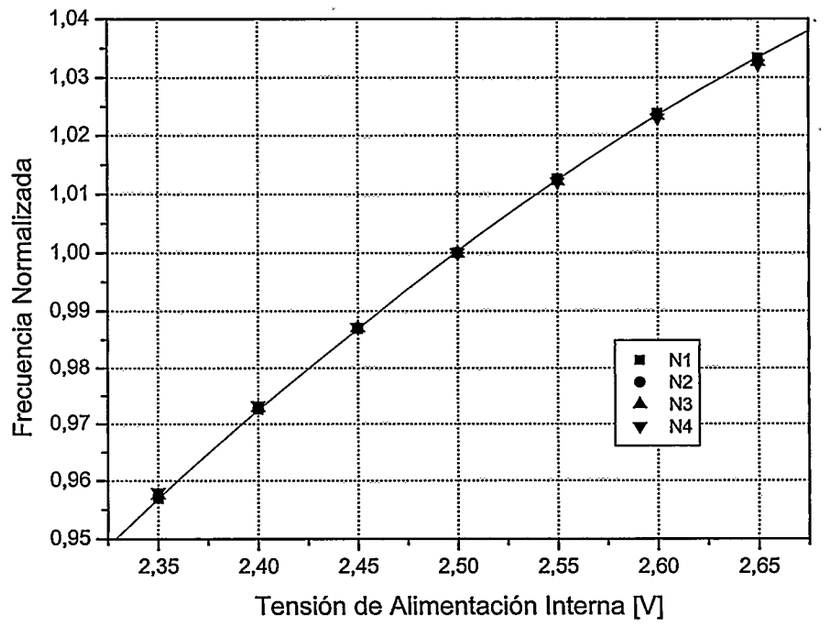



Fig. 92: Frecuencia de salida vs. Vcc en los sensores N1 a N4 (básicos)

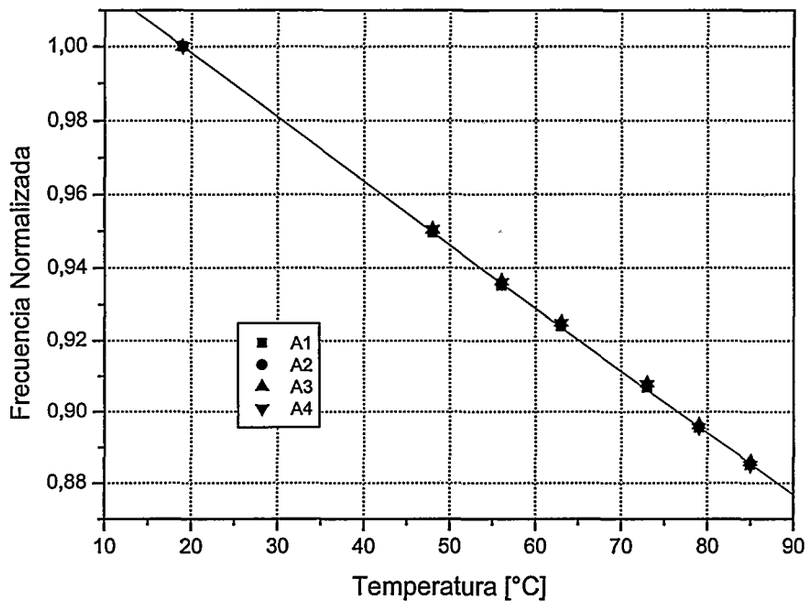



Fig. 93: Frecuencia de salida vs. temperatura en los sensores A1 a A4 (rutado largo)

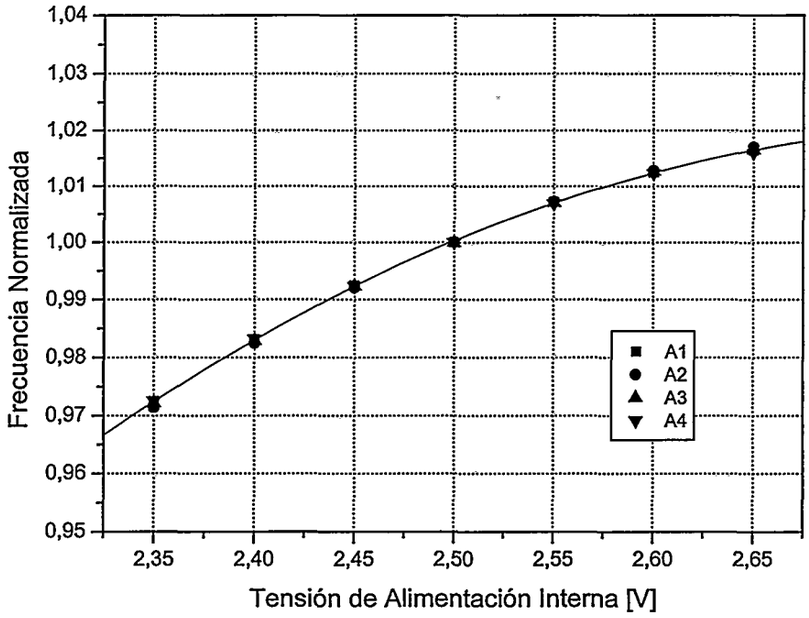



Fig. 94: Frecuencia de salida vs. Vcc en los sensores A1 a A4 (rutado largo)

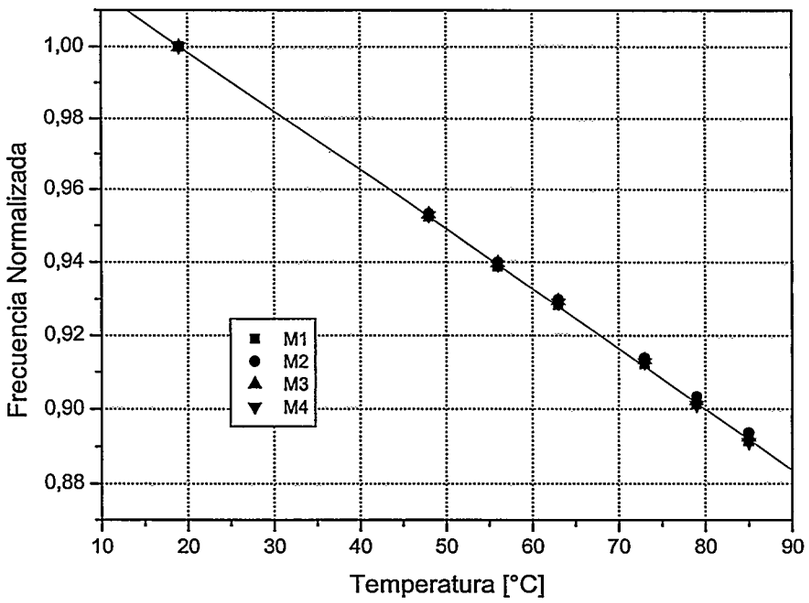



Fig. 95: Frecuencia de salida vs. temperatura en los sensores M1 a M4 (macros, rutado normal)

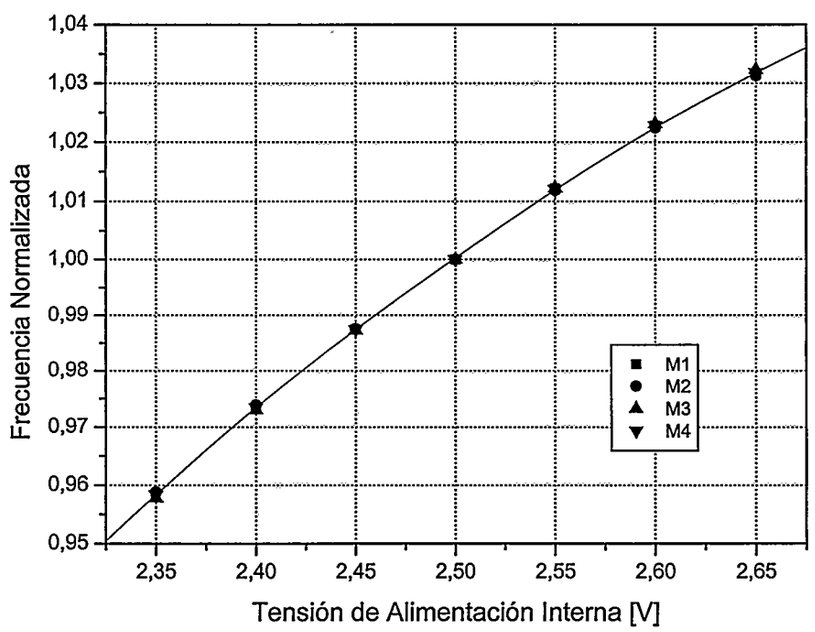



Fig. 96: Frecuencia de salida vs. Vcc en los sensores M1 a M4 (macros, rutado normal)

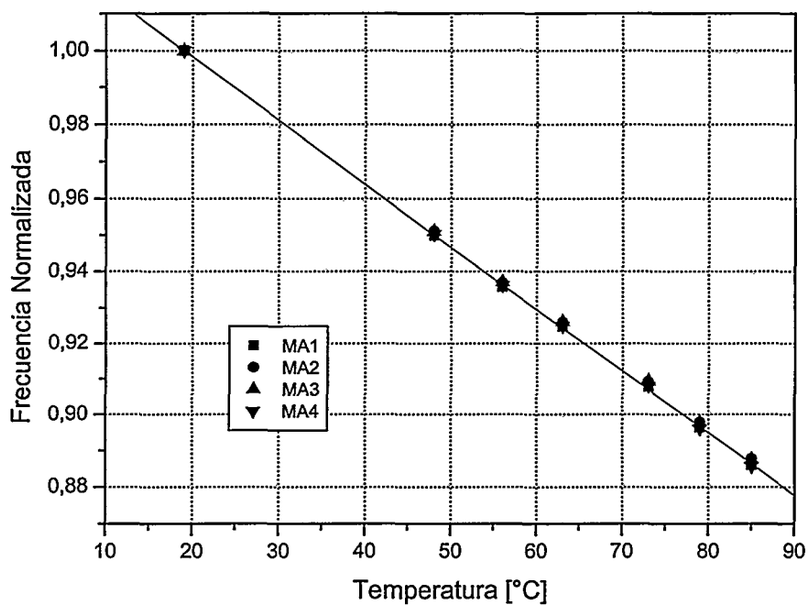



Fig. 97: Frecuencia de salida vs. temperatura en los sensores MA1 a MA4 (macros, rutado largo)

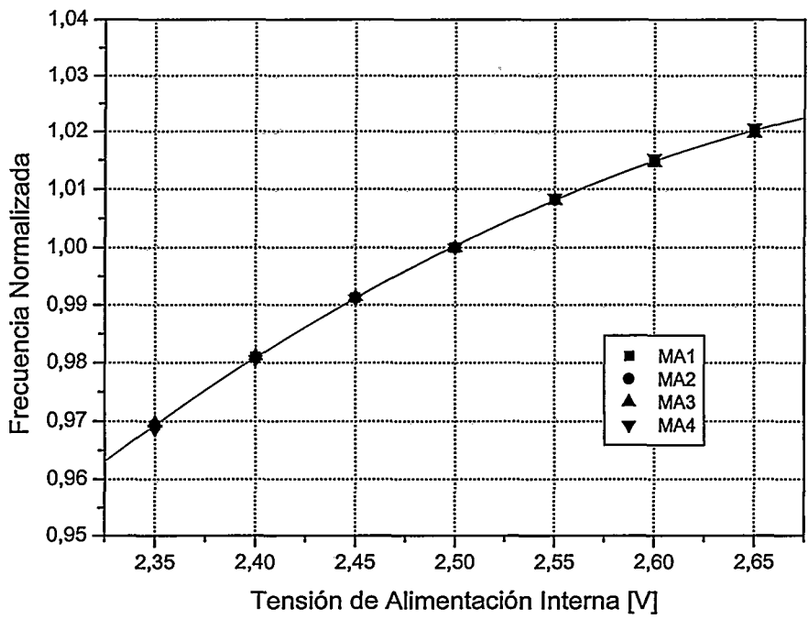



Fig. 98: Frecuencia de salida vs. Vcc en los sensores MA1 a MA4 (macros, rutado largo)

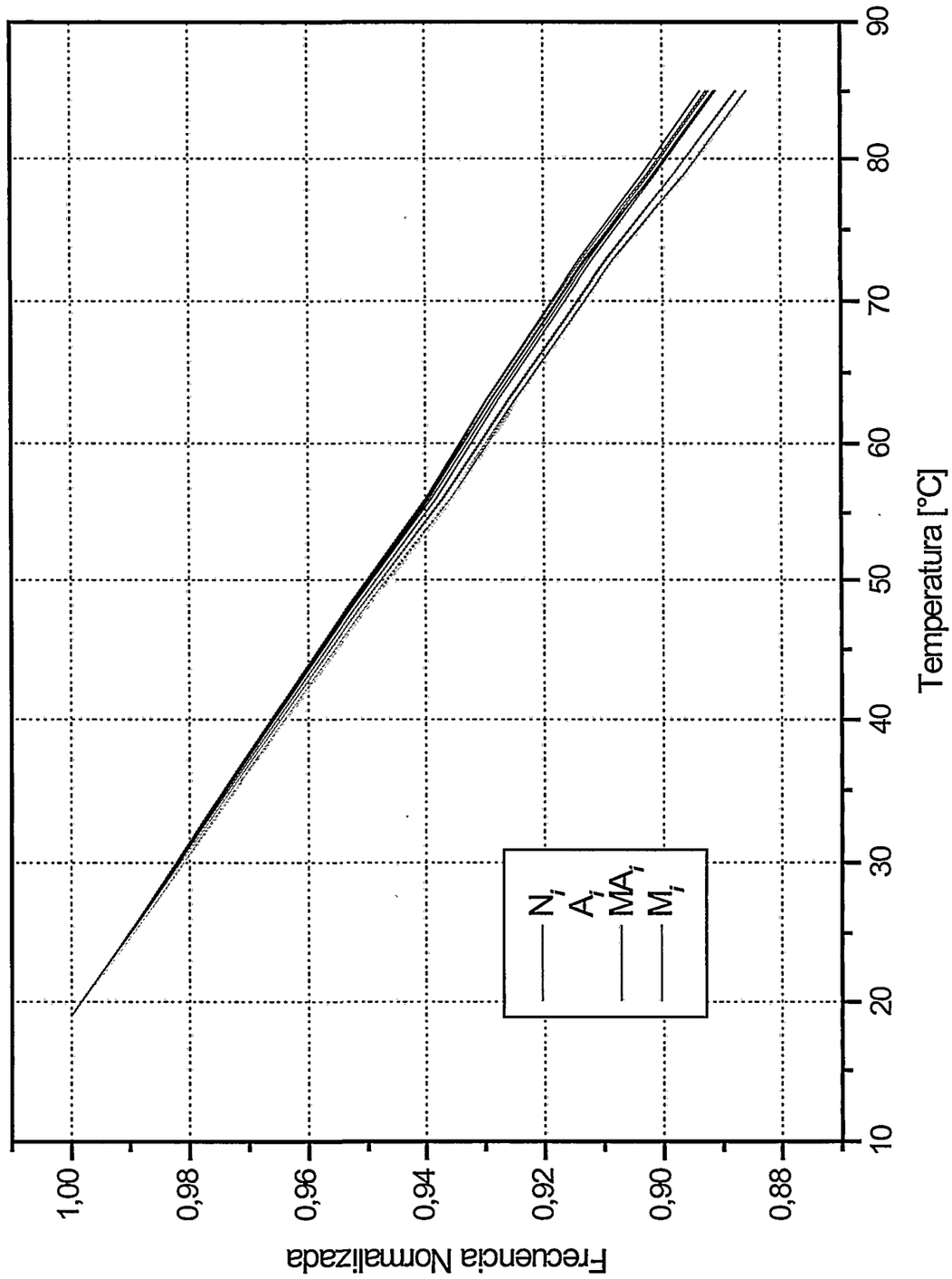



Fig. 99: Frecuencia de salida vs. temperatura para todos los tipos de sensores

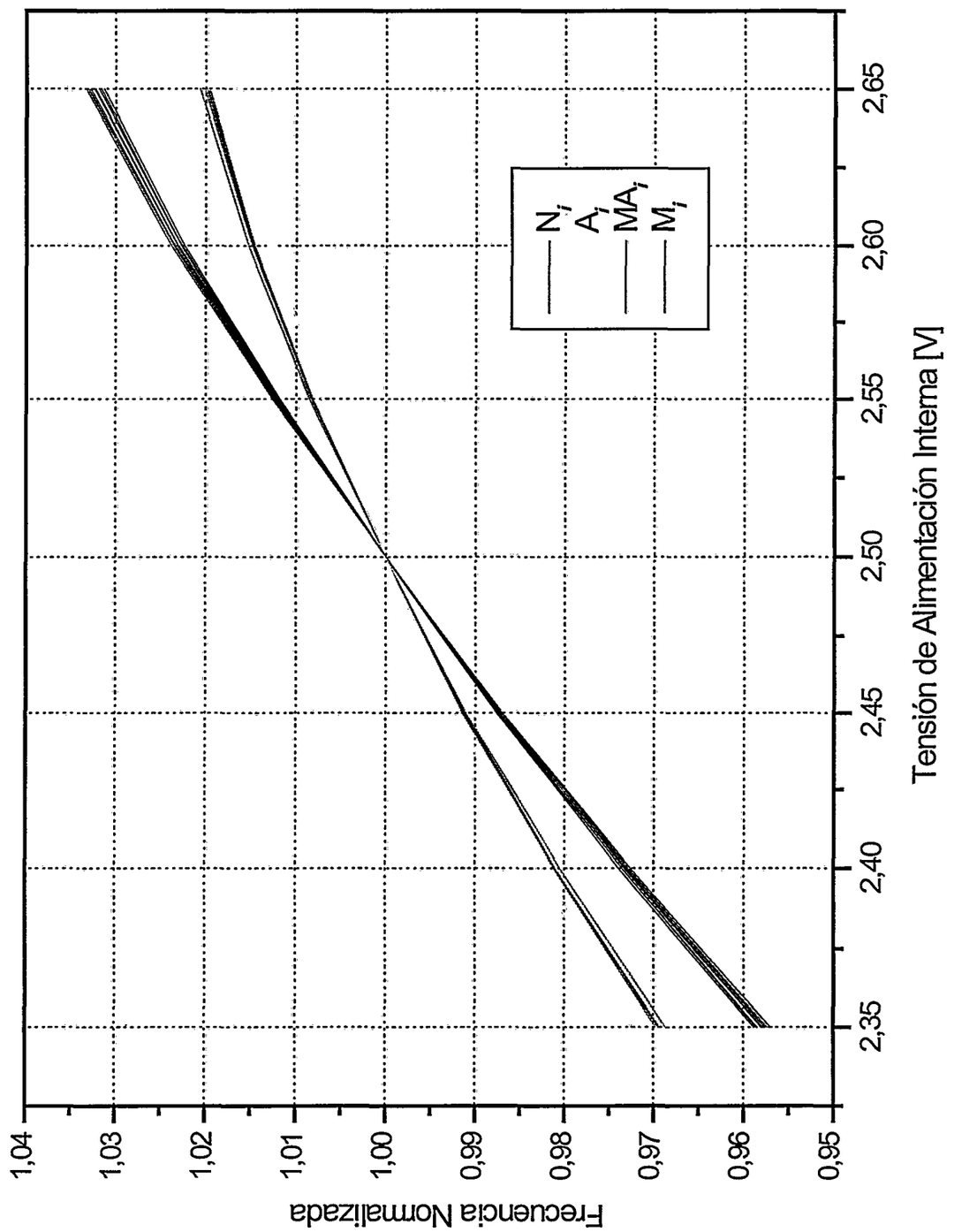



Fig. 100: Frecuencia de salida vs. Vcc para todos los sensores

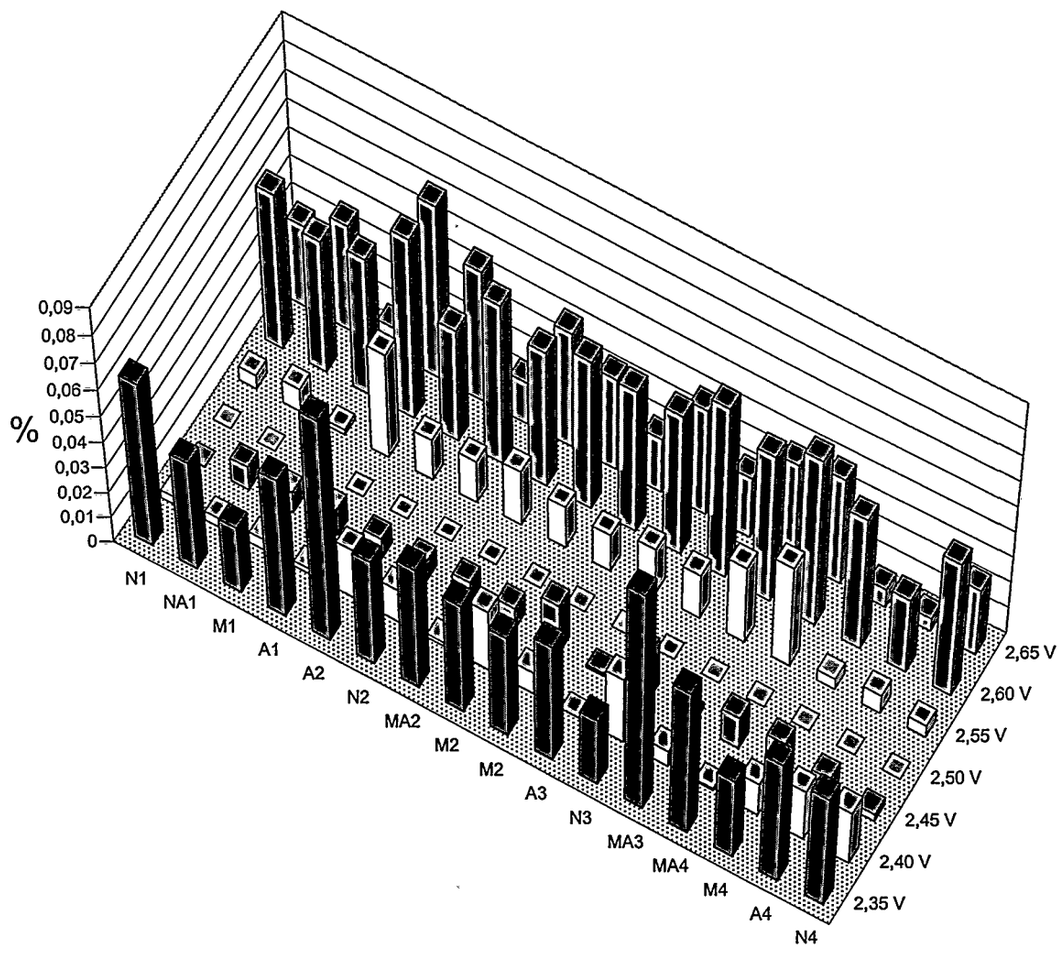



Fig. 101: Relación (en %) entre las variaciones de la frec. normalizada con Vcc a 19 °C y a 85 °C

Todas estas gráficas vuelven a reafirmar las conclusiones obtenidas tanto en el capítulo 4 como en secciones anteriores:

- Los osciladores en anillo tienen una respuesta lineal con respecto a las variaciones de temperatura
- La respuesta frente a las variaciones de Vcc no es tan lineal, ajustándose mejor con una parábola

- Una vez normalizada la frecuencia, los osciladores con el mismo layout tienen una respuesta prácticamente idéntica

Resultan especialmente interesantes la Fig. 99 y la Fig. 100, donde se han dibujado las respuestas de los 16 sensores, cada una coloreada según el tipo de sensor. Por un lado se puede ver que la respuesta en temperatura es muy parecida para todos los sensores, agrupándose ligeramente en dos conjuntos: rutado normal y rutado largo. Con respecto a la respuesta frente a las variaciones de Vcc, claramente cada tipo de sensor tiene su propio comportamiento, siendo muy parecidos en las dos alternativas (JBits y macro) con rutado normal.

Dentro de cada categoría el comportamiento es muy similar; aunque se tomase el mismo coeficiente para todos los sensores de un mismo tipo, el error que se cometería no superaría ±1 °C en el rango medido de 60 °C. Este dato es sin duda positivo, aunque la deriva entre sensores, por muy pequeña que sea, tiene un aspecto negativo: puede ser un grave problema a la hora de hacer mapas térmicos del dispositivo cuando su temperatura es muy elevada. Para estas aplicaciones se debería desarrollar algún mecanismo de calibración que compensase estas diferencias en las respuestas.

Por último, la Fig. 101 trata de evaluar si el comportamiento frente a variaciones de Vcc varía con la temperatura. Aunque ya se puede intuir que no, porque si fuera así los resultados obtenidos en el punto 7 no hubieran sido tan buenos. En cualquier caso, para comprobarlo se ha graficado el valor absoluto de las diferencias porcentuales entre el comportamiento a 19 °C y a 85 °C, según esta fórmula:

$$100 \cdot \left| \frac{FrecNorm_{19°C}}{FrecNorm_{85°C}} - 1 \right|$$

Como se puede ver en la figura, las diferencias son mínimas; probablemente vengan incluso causadas por errores en la medida. Similares resultados se obtuvieron con las medidas a 73 °C; por lo tanto se puede afirmar que la respuesta frente a Vcc no se ve alterada por la temperatura.

# 10. Conclusiones

La primera conclusión de este capítulo es que la medida de temperatura es una aplicación muy adecuada para ser manejada a través de la reconfiguración en tiempo de ejecución (RTR):

- Las constantes térmicas hacen que la temperatura varíe lentamente, al menos más despacio de lo que se tarda en reconfigurar la FPGA. Es entonces factible insertar dinámicamente el sensor cuando se quiera hacer la medida; en el tiempo en que tarde en cargarse no variará significativamente la temperatura.
- La temperatura no se mide continuamente, sino que se hace en intervalos relativamente largos, y cada medida tarda unos pocos $\mu$s en realizarse. ¿Por qué gastar entonces área en una aplicación que está activa mucho menos del 1% del tiempo? Se puede dejar el área libre para otras aplicaciones; insertar el sensor sólo cuando queramos hacer la medida, y luego eliminarlo.
- Aunque normalmente sólo se medirá la temperatura en punto, y el gasto de área por dejar un sensor continuamente en la FPGA no es muy grande. Pero si el valor obtenido se sale fuera de los límites seguros de operación, se insertará dinámicamente una batería de sensores en el chip, que tratarán de detectar la causa de ese incremento de temperatura. Si no se emplease RTR y esa batería de sensores estuviera continuamente en la configuración, implicaría un gasto de área que puede ser inaceptable.

Estas conclusiones no sirven de nada si no es posible crear un sensor que pueda ser operador mediante RTR; pero en este capítulo se ha demostrado que esto si que es posible:

- Empleando JBits se ha diseñado un sensor para FPGAs de la familia Virtex, basado en osciladores en anillo. JBits es una de las pocas herramientas (y la más empleada) que permite diseñar para RTR. El sensor es de pequeño tamaño (8 x 2 CLBs), y puede ser insertado o eliminado dinámicamente de la configuración.
- El sensor está totalmente autocontenido; no necesita de ningún componente ni señal externa, salvo el reloj del sistema. Además, se maneja completamente a

través del puerto de configuración, a través de él se inserta en la FPGA, se activa, se lee su resultado (mediante readback), y se elimina de la configuración para dejar espacio a otros circuitos.

- Los experimentos indican que, como otros osciladores en anillo, tiene una respuesta lineal frente a la temperatura, con una buena sensibilidad. Y además, el comportamiento del sensor, una vez normalizado, es independiente de la posición en la FPGA o incluso del dispositivo que se use.

Sin embargo, uno de los problemas de los osciladores en anillo es su sensibilidad frente a las variaciones en la tensión de alimentación. Pero en este capítulo se ha demostrado que es posible crear una pareja de sensores con distinta sensibilidad respecto a Vcc con sólo variar el rutado. Y a partir de la información de ambos sensores, obtener con una precisión muy buena (±1°C, ±5mV) tanto la temperatura como la tensión de operación.

Aunque la RTR sea una buena idea, lo cierto es que hoy por hoy se utiliza sólo en aplicaciones de investigación, y en el mundo real se siguen utilizando las herramientas clásicas. Sería una lástima que no se pudieran utilizar parte de las ventajas del sensor JBits en diseños convencionales, en especial el tener un sensor autocontenido, que no gaste patas de E/S y que se pueda manejar a través del puerto de configuración de una manera transparente. En este capítulo se ha demostrado que esto es posible empleando macros físicas, y que, gracias a las características de la reconfiguración parcial de Virtex, la operación del sensor no interfiere en el funcionamiento de otros circuitos que pudiera haber en la FPGA.

# Capítulo 6.

# Mapas térmicos en FPGAs

En los capítulos anteriores se han presentado una serie de sensores de temperatura que ocupan un área mínima del chip, y que pueden ser colocados en cualquier punto de él. Estas características abren el camino a la realización un mapa térmico de una FPGA en funcionamiento, una aplicación inédita en sistemas reconfigurables. Esta posibilidad es muy atractiva, pues es la única manera de saber que parte del circuito es la que está consumiendo más. Para comenzar, se debe comprobar si es posible construir estos mapas térmicos sin alterar el circuito que se quiere medir. Además, verificar que los consumos localizados de potencia en el chip crean gradientes de temperatura suficientemente grandes como para que puedan ser observados por los sensores. Todo esto se evalúa positivamente en los experimentos que se describirán a continuación.

## 1. Experimentos basados en puntos calientes

La primera prueba que se ideó para evaluar la utilidad de los mapas térmicos fue mover un circuito con un consumo puntual y relativamente elevado (un punto caliente) por todo el dispositivo. Si todo iba bien, debería haber una correlación clara entre la localización del punto y el área de mayor temperatura en la FPGA. Para realizar estos experimentos se escogió una FPGA de tamaño moderado, la XCV800HQ240-4. Como sensores se utilizaron los descritos en el capítulo anterior, porque tienen la ventaja de ser completamente autocontenidos, no necesitan de ningún equipo externo de medida. Por lo demás, la preparación de los experimentos fue completamente similar a la descrita en el capítulo anterior. Se utilizó la placa de prueba Xilinx AFX, cuyas ventajas en este tipo de experimentos ya se han mencionado: al no tener ningún componente adicional, el

consumo de potencia puede ser medido sin mayores problemas, y al mismo tiempo se evita cualquier tipo de interferencia en la medida de la temperatura.

Los experimentos se realizaron en el entorno más estable al que se pudo tener acceso: la sala blanca de la UAM, donde las variaciones de temperatura ambiente son menores de 1 °C.

Como ya se comentó en el capítulo anterior, este sensor se ha diseñado con JBits, que se utiliza también para controlarlo y leer su respuesta. La interfaz con JBits se realizó a través del puerto paralelo del PC, utilizando el hardware diseñado a medida descrito en el apéndice B.

En total se mapearon en la FPGA un total de 40 sensores, dispuestos en una matriz de 4 filas por 10 columnas. Esto fue algo muy sencillo de realizar, pues al estar diseñado el sensor como un *RTPCore*, sólo hubo que instanciarlo tantas veces como haga falta:

```
for (i=0; i<numOfSensors; i++)
{
      sensor[i] = new TemperatureSensor("Sensor", clk);
}
for (i=0; i<numOfSensors; i++)
{
      sensorOffset[i] = sensor[i].getRelativeOffset();
      sensorOffset[i].setHorOffset(Gran.CLB, sensorXPos[i]);
      sensorOffset[i].setVerOffset(Gran.CLB, sensorYPos[i]);
}
for (i=0; i<numOfSensors; i++)
{
     sensor[i].implement(TimeEnable,RingEnable,CountEnable);
}
Bitstream.connect(clk);
```

Estos sensores se utilizaron para medir una serie de 15 circuitos, que implementaban un punto caliente en distintas posiciones de la FPGA. Los sensores se añadieron directamente sobre el *bitstream* del circuito empleando JBits. Por supuesto, al diseñar el circuito del punto caliente ya se había asegurado previamente que quedaba libre el área que iban a ocupar los sensores.

Los puntos calientes fueron diseñados a partir de esquemáticos utilizando las herramientas convencionales (Foundation 3.1). Básicamente, están compuestos por una matriz de 4 x 4 CLBs en la que todas sus LUTs y sus flip-flops están conmutando a la mitad de la frecuencia de reloj. O sea, un total de 64 LUTs de 4 entradas y flip-flops tipo D. La conmutación comienza en el primer flip-flop, que está realimentado a través de un inversor (circuito de divisor de frecuencia ÷2), y luego se propaga al resto de flip-flops. Antes de cada flip-flop la señal pasa por un inversor; así, junto con un símbolo FMAP, se consigue forzar el uso de las tablas de *lookup*. El objetivo de emplear las LUTs es tratar de incrementar el consumo.

Adicionalmente, la conmutación se habilitaba o deshabilitaba desde un pin externo; esto es porque realmente el primer flip-flop no tiene un inversor conectado, sino una puerta NAND. Si este pin está a 1, entonces funciona como divisor de frecuencia, y si no, se queda siempre a 1, con lo que se deshabilita la conmutación en toda la cadena, quedando desactivado el punto caliente. En realidad, en los experimentos esta activación se hizo por un mecanismo ligeramente distinto: actuando con JBits directamente sobre los contenidos de la LUT del primer flip-flop. De esta manera se conseguía poder controlar el punto caliente desde JBits, sin tener que utilizar ningún otro circuito externo que actuase sobre la pata de control.

Para agrupar todos estos elementos en un matriz de 4 x 4 CLBs se utilizaron directivas de emplazamiento relativo, de tal manera que una única directiva de emplazamiento absoluto es la que fija la posición de el punto caliente en la FPGA.

Por último, los circuitos utilizan un DLL para duplicar la frecuencia de reloj, y necesitan además implementar la celda de *readback* (necesaria para poder leer el valor devuelto por los sensores de temperatura).

En la siguiente figura se muestra el esquemático de estos circuitos; es igual para los 15 restantes, sólo varía la directiva de emplazamiento absoluto que fija su posición en la

FPGA. Dado lo grande que es el esquemático, sólo se muestra la mitad que se corresponde con el *slice* 0; la otra mitad es exactamente igual, con la salvedad de que las directivas de emplazado relativo llevan el sufijo .S1.

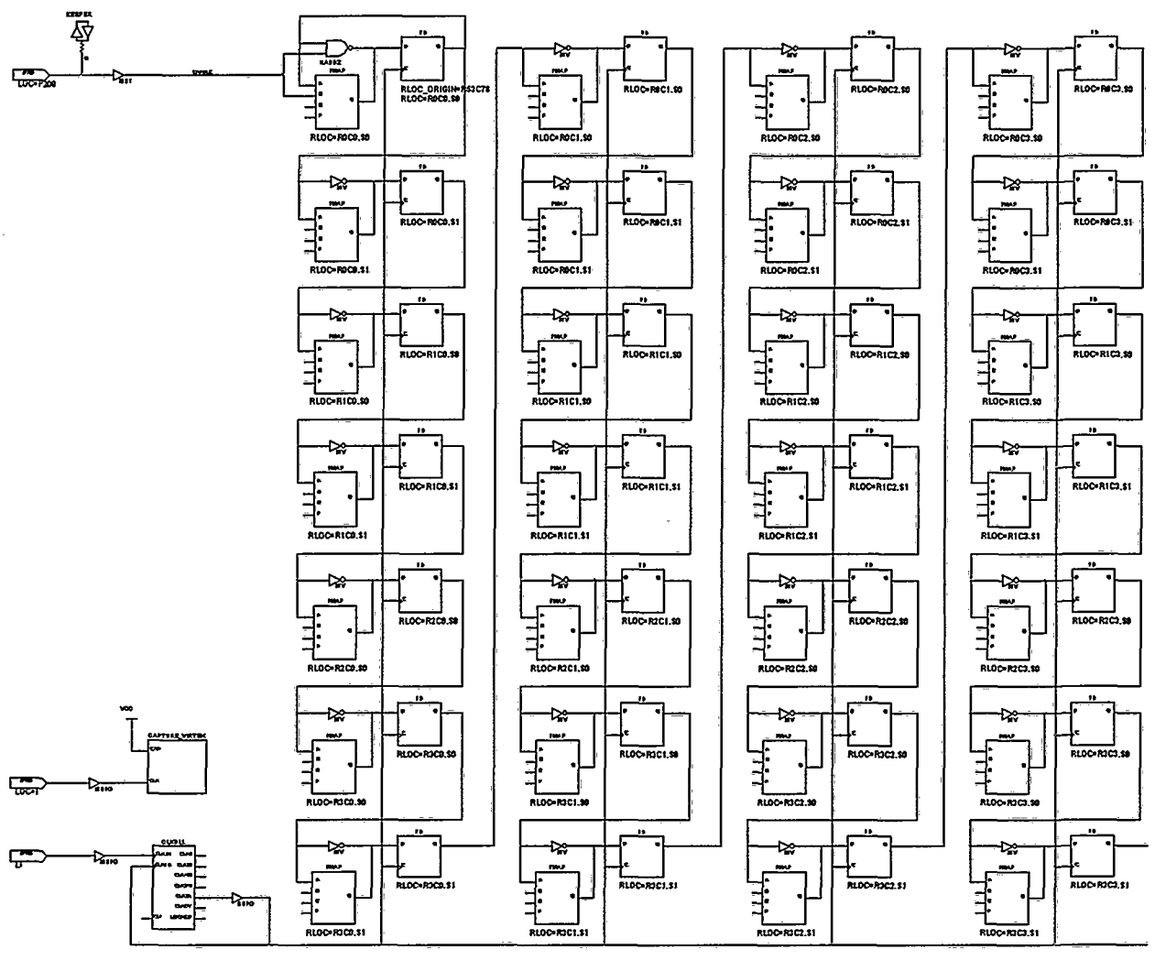


Fig. 102: Esquemático (mitad) del circuito para crear un punto caliente en la FPGA

En la siguiente página se muestra el layout de la matriz de sensores, y a continuación, un diagrama con la posición del punto caliente en los 15 circuitos y su relación con la posición de los sensores.

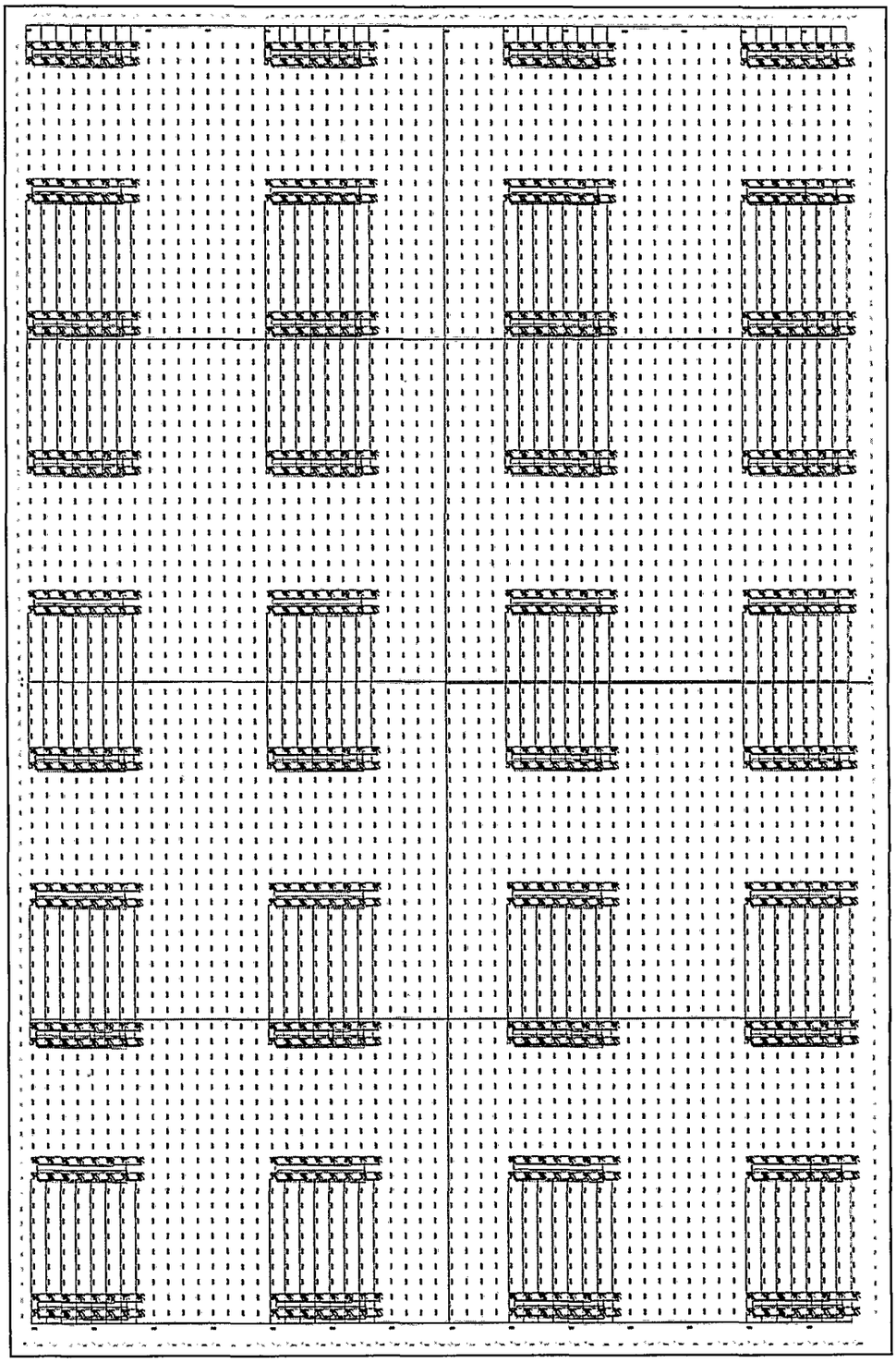


Fig. 103: Layout de la matriz de sensores empleada para crear los mapas térmicos

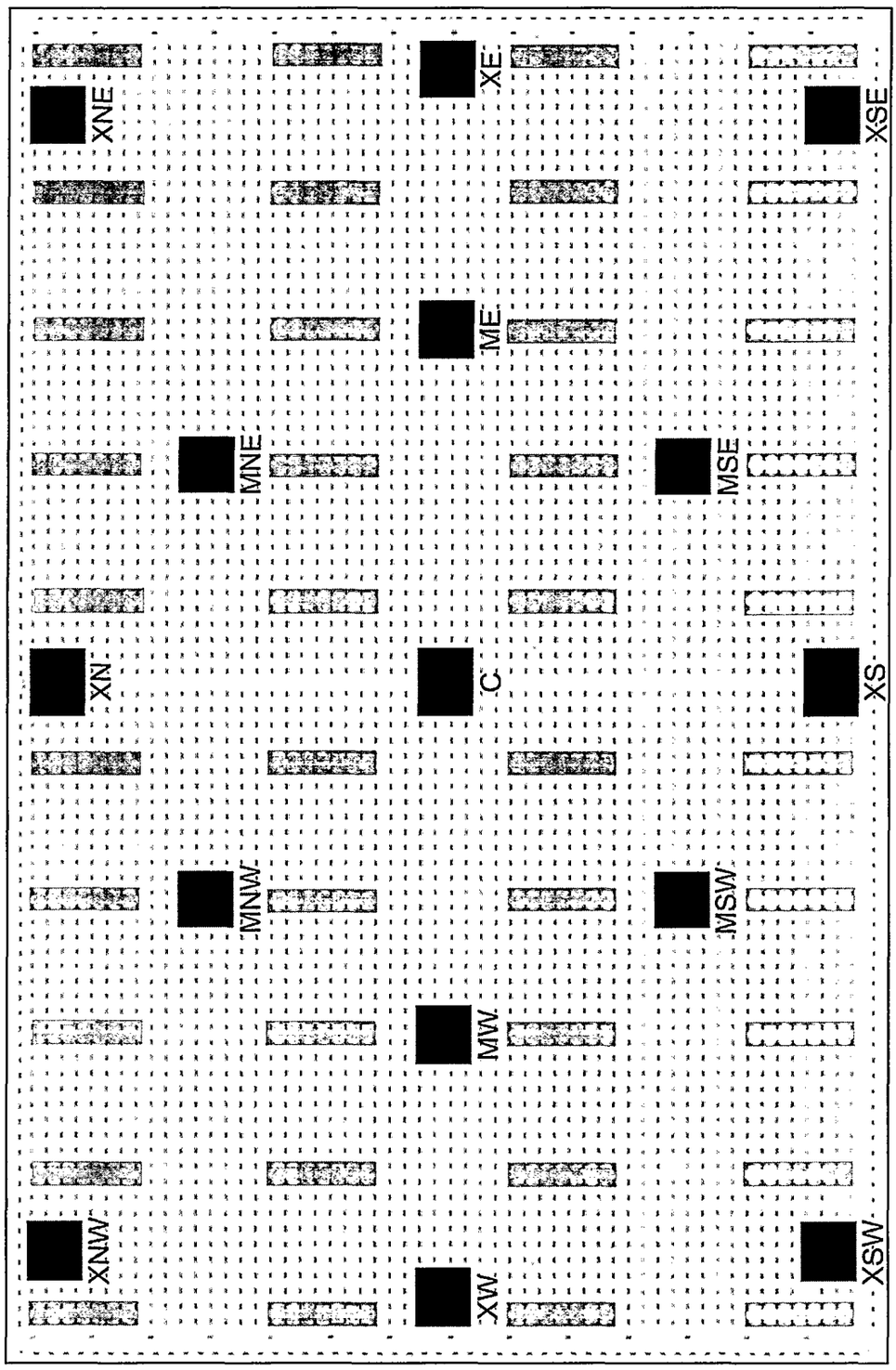



Fig. 104: Localización y nomenclatura de los puntos calientes en la FPGA

Cada circuito tiene un nombre que se corresponde con la posición geográfica del punto caliente en la FPGA. En la siguiente tabla se detallan las posiciones base de los puntos calientes para cada uno de estos 15 circuitos, utilizando el sistema de coordenadas de las herramientas convencionales (no el de JBits) y teniendo en cuenta que la referencia de los puntos calientes está en su esquina superior izquierda.

| Circuito | Fila | Columna |
|---|---|---|
| XNW | 1 | 4 |
| XN | 1 | 41 |
| XNE | 1 | 78 |
| MNW | 10 | 27 |
| MNE | 10 | 55 |
| XW | 27 | 1 |
| MW | 27 | 18 |
| C | 27 | 41 |
| ME | 27 | 64 |
| XE | 27 | 81 |
| MSW | 43 | 27 |
| MSE | 43 | 55 |
| XSW | 53 | 4 |
| XS | 53 | 41 |
| XSE | 53 | 78 |

Tabla 17: Coordenadas de los puntos calientes en la FPGA

Respecto a las posiciones de los sensores, se han colocado equiespaciados en el área de CLBs de la FPGA, dejando un espaciado entre ellos de 8 CLBs en vertical y 7 CLBs en horizontal, salvo en las dos columnas a ambos lados del centro, que están espaciadas 8 CLBs.

## 1.1. Ejecución de los experimentos

Para cada uno de los circuitos se hicieron dos medidas, una a 50 MHz, en la que el punto caliente disipaba 50 mW, y otra a 100 MHz, con un consumo de 100 mW. Este último caso estaba fuera de los márgenes recomendados de funcionamiento de la FPGA, pues sólo se garantiza que el DLL puede duplicar la frecuencia hasta 90 MHz. Pero como se comprobó que los circuitos funcionaban correctamente, se optó por seguir adelante con los experimentos.

Para cada circuito se hizo un test automático en tres tiempos: en el primero el punto caliente está activado, se desactiva en el intermedio, y se vuelve a habilitar en el último. La duración total del experimento era de 5000 s, y cada 10 s se construía un nuevo mapa térmico. En la Fig. 105 y en la Fig. 106 se puede ver el efecto que provoca la activación del circuito "XNW" (en una esquina del dado) en la frecuencia de salida del sensor más próximo y más lejano a él. La Fig. 105 se corresponde con un consumo del punto caliente de 50 mW, y la Fig. 106 de 100 mW. En estas figuras se ve como la activación del punto caliente se traduce en un incremento de la temperatura en todos los puntos del chip, más acusado cuanto más próximo se encuentre el sensor. Se pueden comprobar también lo grandes que son las constantes térmicas dominantes. Probablemente a esto ayude el zócalo sobre el que va montada la FPGA en la placa Xilinx AFX. Como se puede comprobar en la Fig. 105, hay veces en que el punto de partida de la frecuencia no es igual al de llegada, aunque teóricamente no debería ser así. Esto es debido a la variación de la temperatura ambiente durante los experimentos. Como ya se comentó, se trató de buscar la sala con la temperatura más estable, pero los resultados muestran que hay que ser más exigente con el entorno. Una tanda de experimentos más rigurosos debería hacerse en una cámara climática de alta calidad, cuya temperatura tuviese variaciones de menos de 0,1 °C.

Por último, desde la Fig. 107 a la Fig. 118 se muestran los resultados de los experimentos para los circuitos más representativos. No se muestran temperaturas absolutas, sino incrementos de temperatura al activar el punto caliente. Esto es así porque la temperatura absoluta no es tan visual, pues está dominada por elementos de mucho mayor consumo, como el DLL.

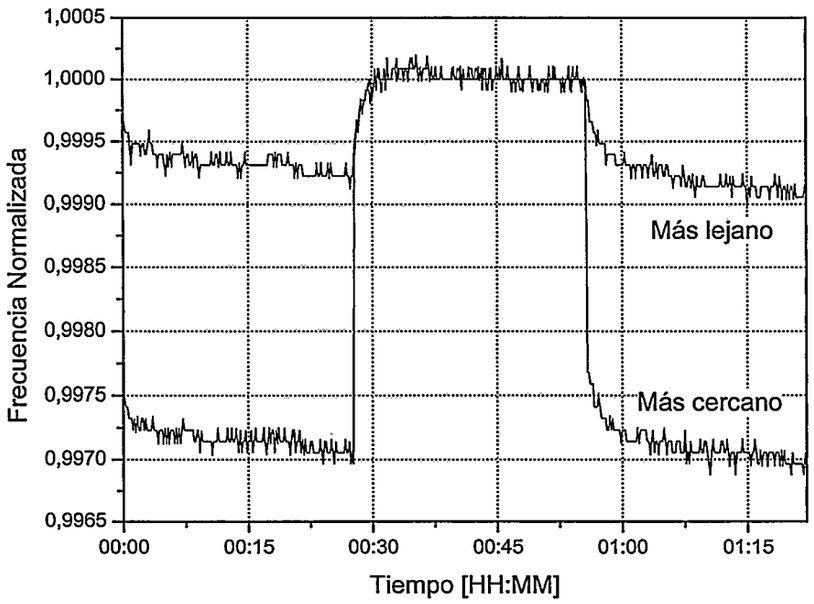



Fig. 105: Efecto del punto caliente "XNW" de 50 mW en los sensores más próximo y más lejano

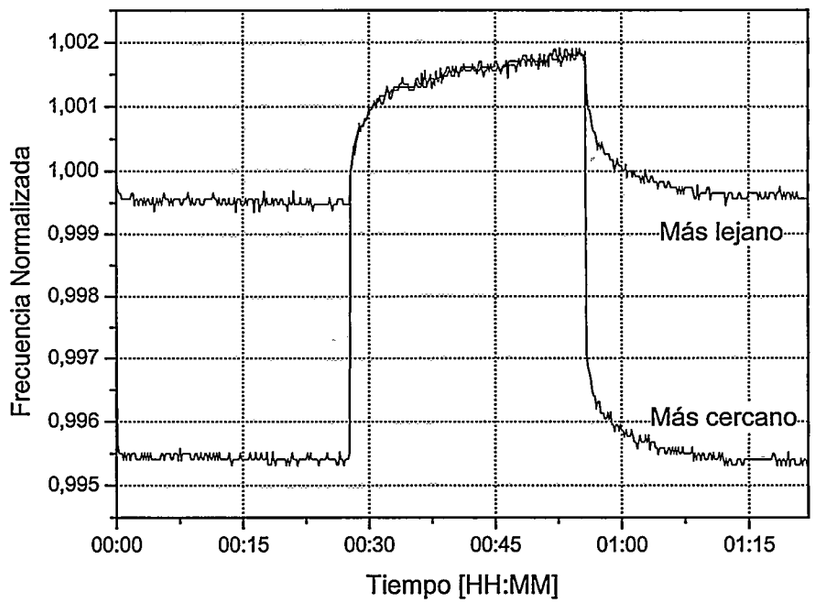



Fig. 106: Efecto del punto caliente "XNW" de 100 mW en los sensores más próximo y más lejano

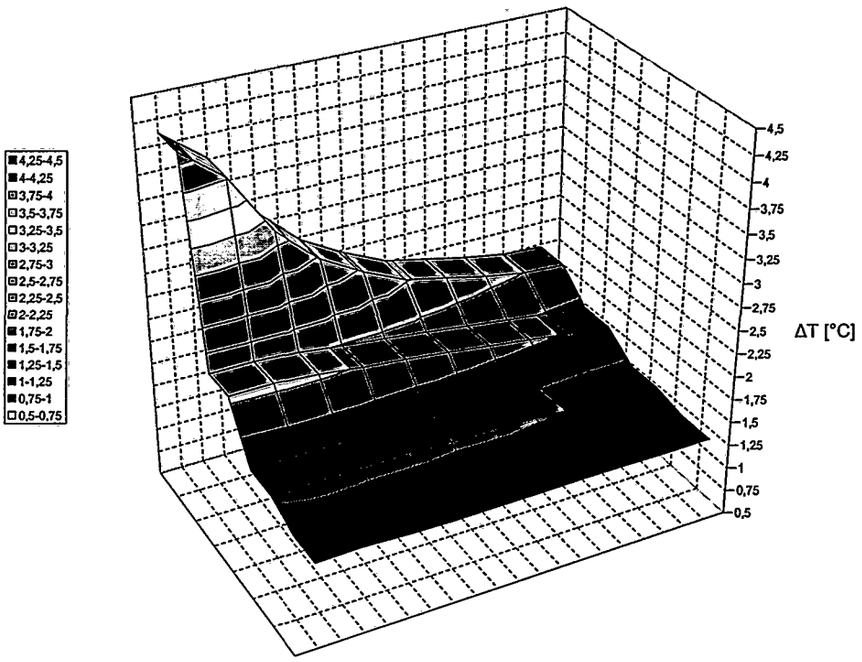



Fig. 107: Gradientes de temperatura al activar el punto caliente "XNW" de 100 mW

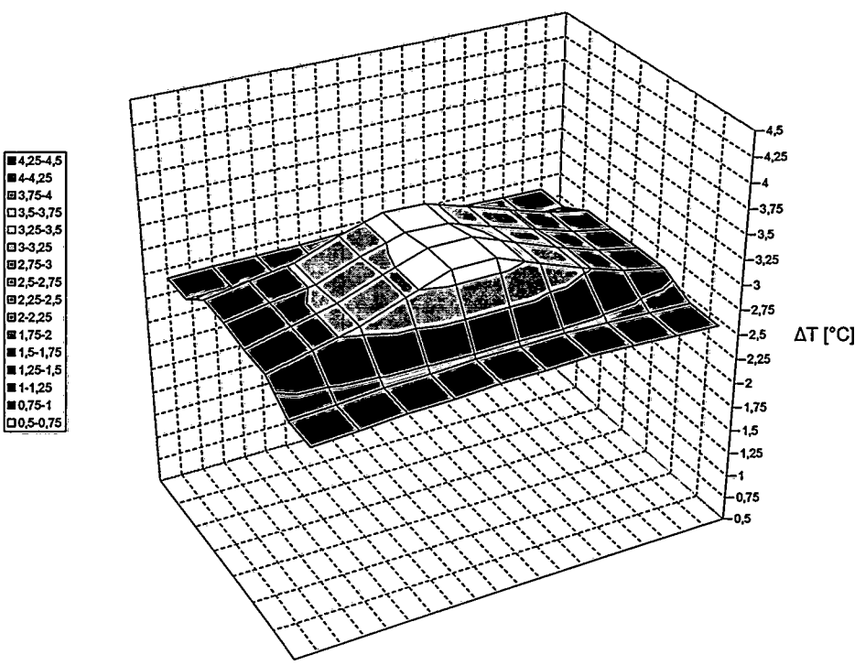



Fig. 108: Gradientes de temperatura al activar el punto caliente "C" de 100 mW

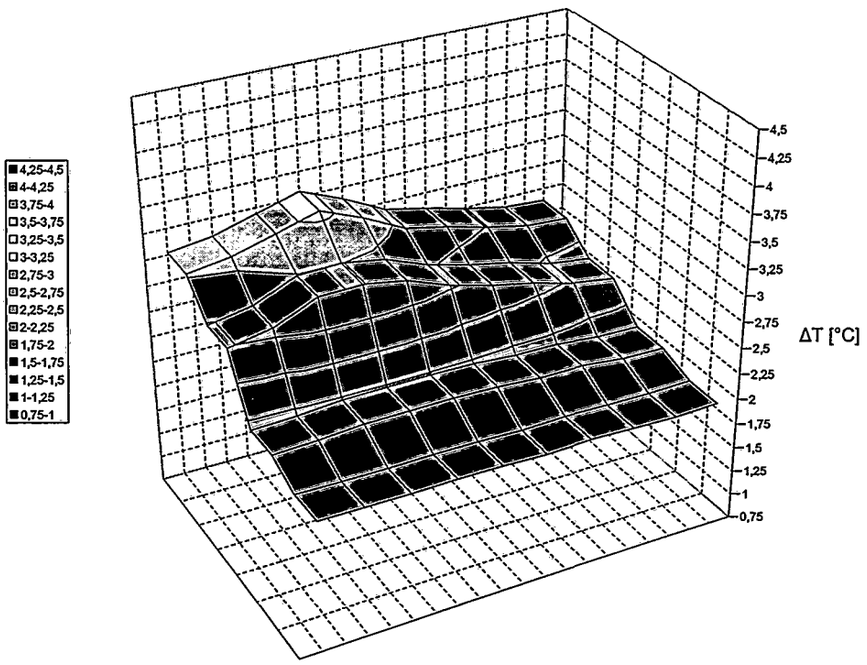



Fig. 109: Gradientes de temperatura al activar el punto caliente "MNW" de 100 mW

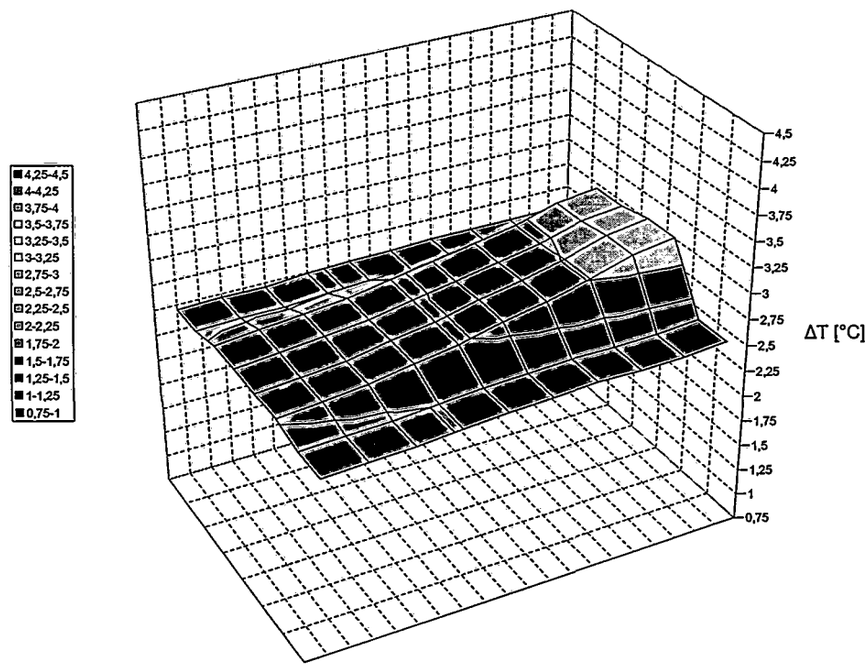



Fig. 110: Gradientes de temperatura al activar el punto caliente "XE" de 100 mW

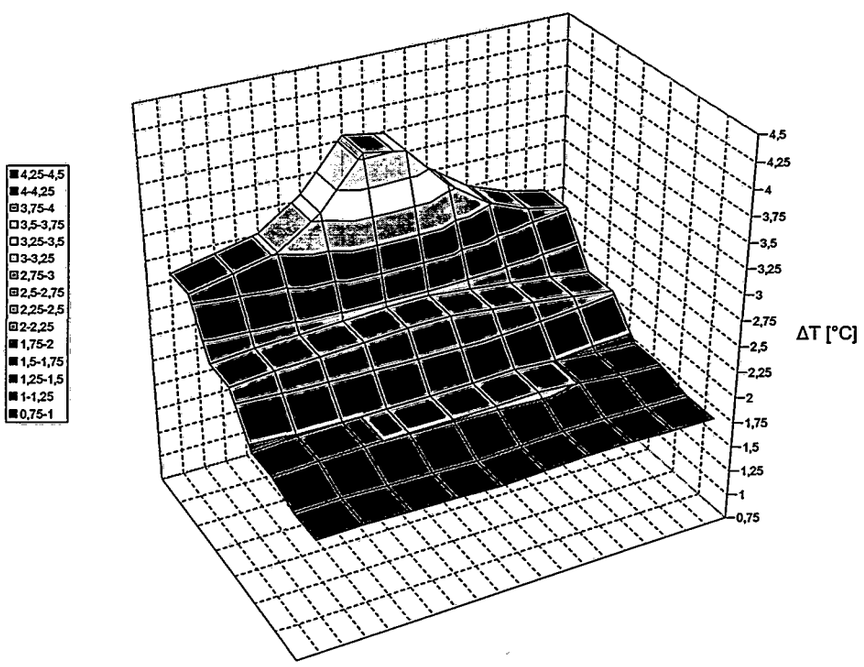



Fig. 111: Gradientes de temperatura al activar el punto caliente "XN" de 100 mW

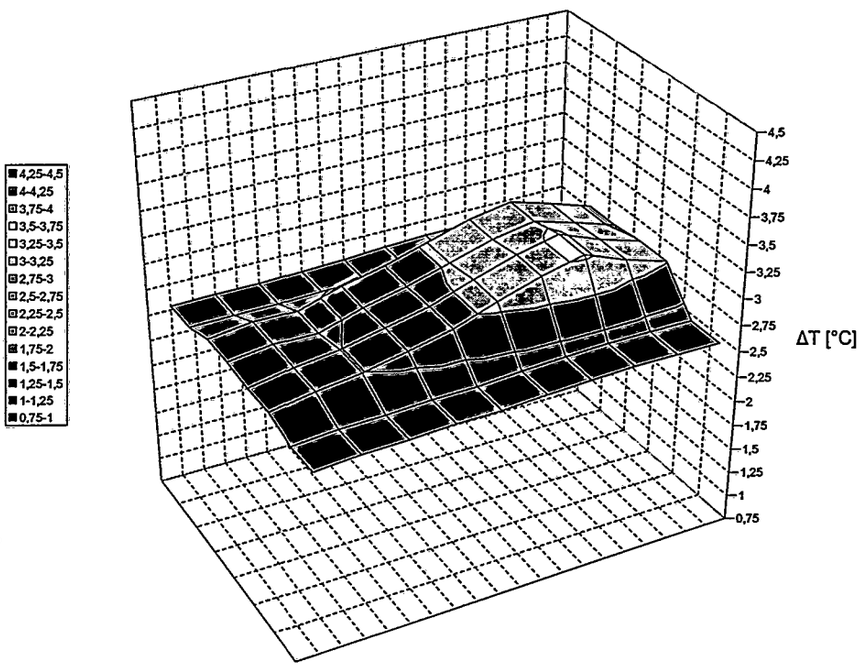



Fig. 112: Gradientes de temperatura al activar el punto caliente "ME" de 100 mW

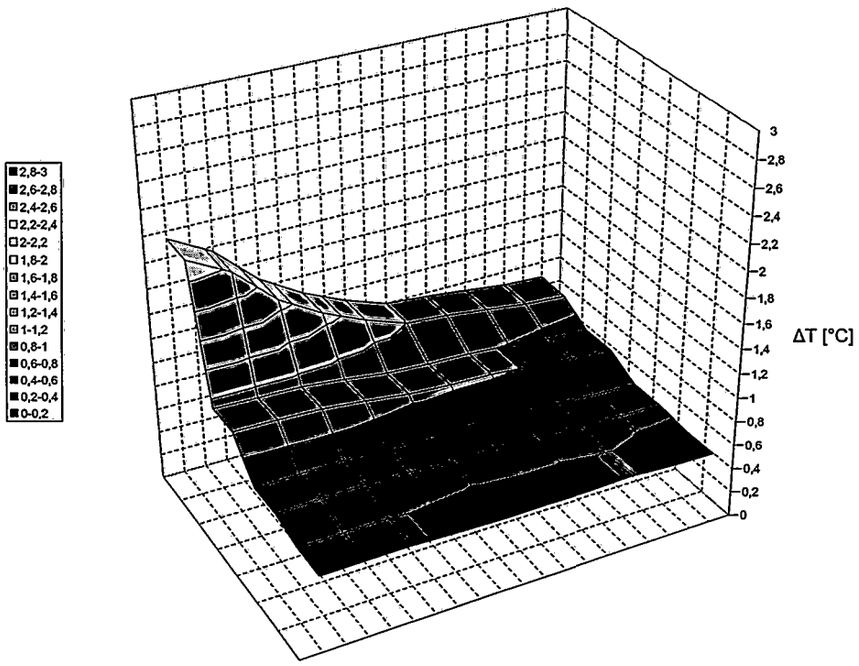



Fig. 113: Gradientes de temperatura al activar el punto caliente "XNW" de 50 mW

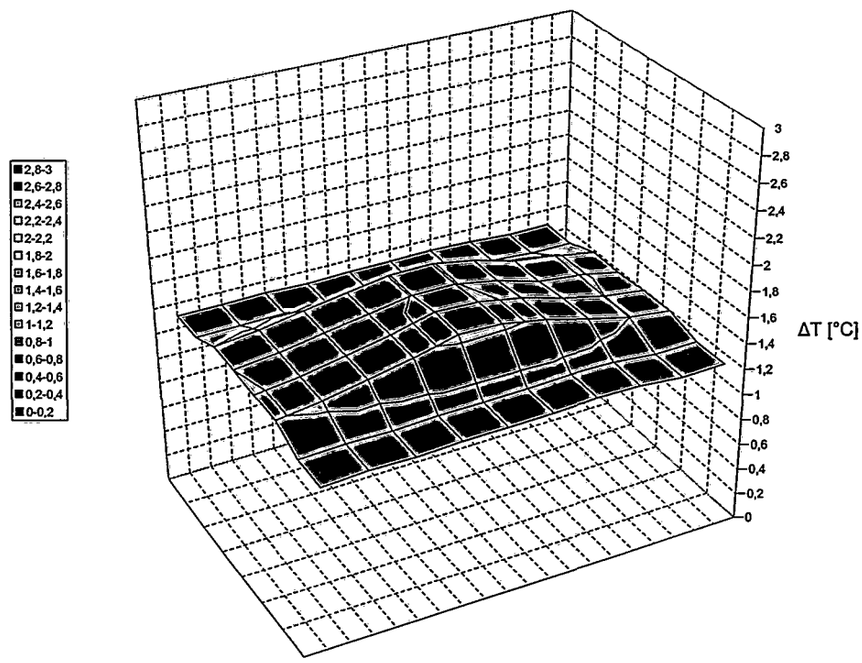



Fig. 114: : Gradientes de temperatura al activar el punto caliente "C" de 50 mW

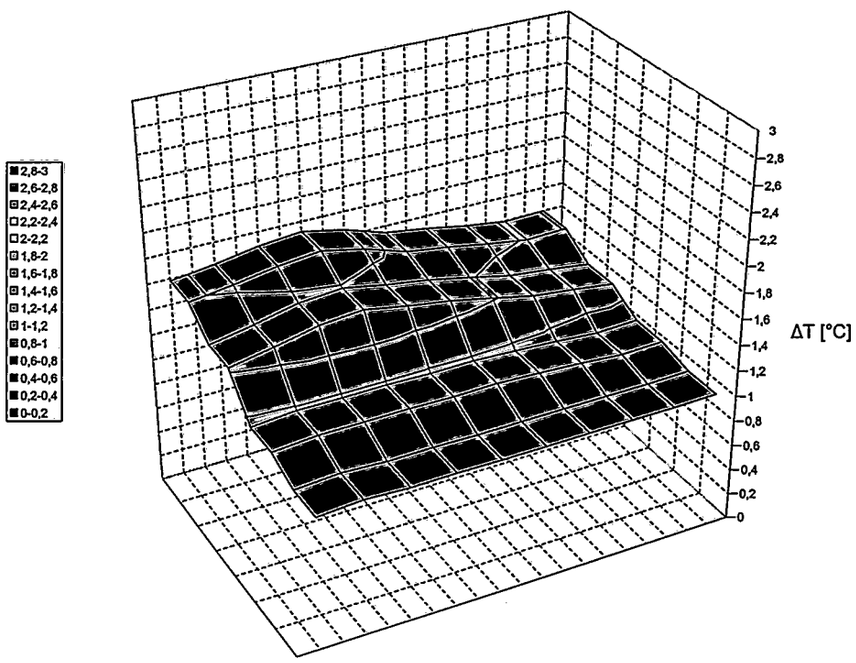



Fig. 115: : Gradientes de temperatura al activar el punto caliente "MNW" de 50 mW

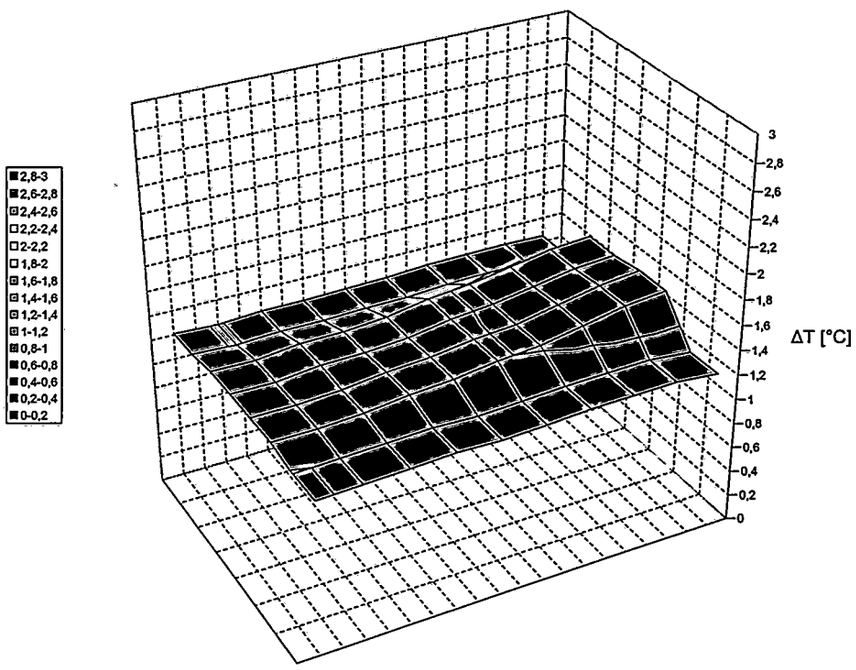



Fig. 116: Gradientes de temperatura al activar el punto caliente "XE" de 50 mW

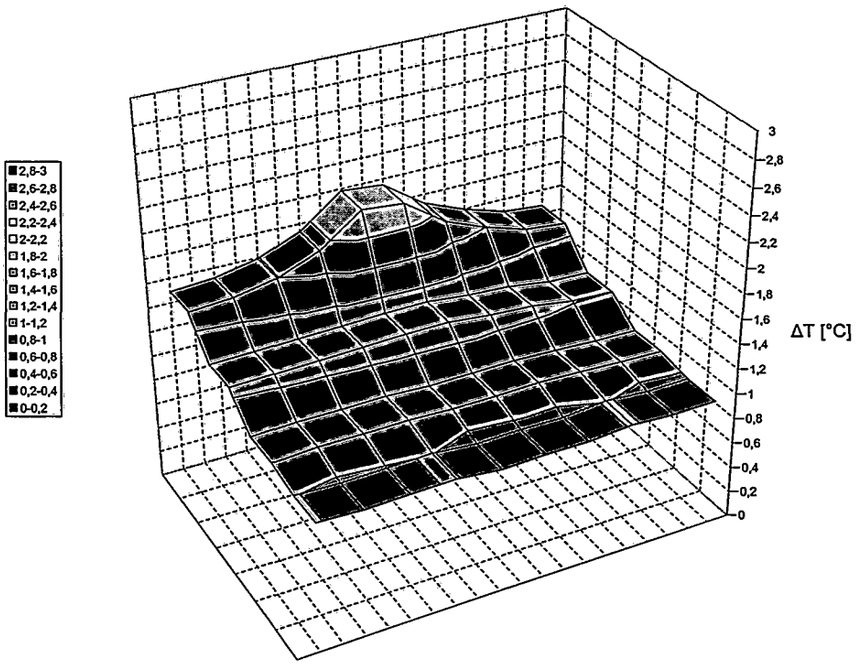



Fig. 117: Gradientes de temperatura al activar el punto caliente "XN" de 50 mW

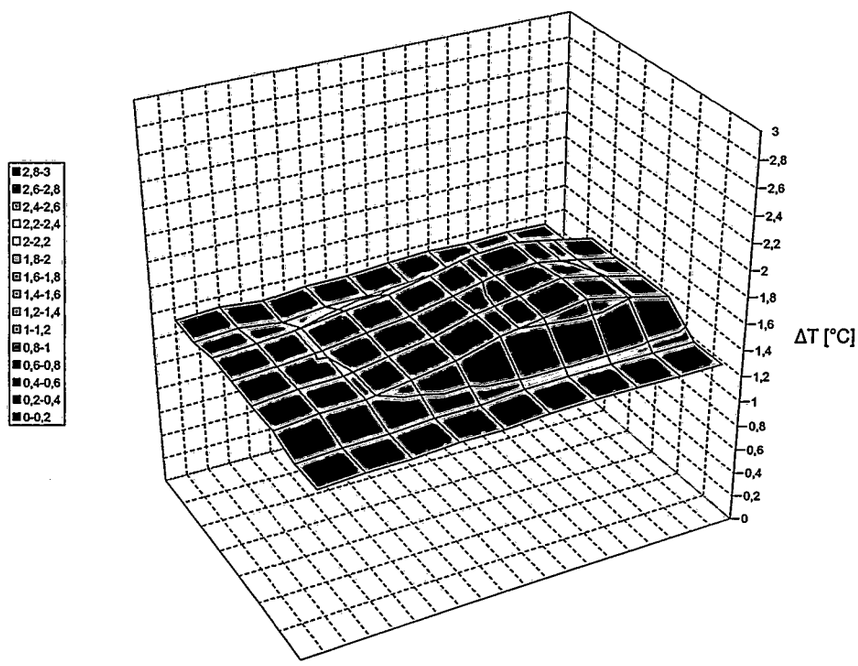



Fig. 118: Gradientes de temperatura al activar el punto caliente "ME" de 50 mW

## 1.2. Resultados

Como se puede ver en todas las figuras anteriores, los resultados han sido muy positivos. No sólo se ha conseguido realizar los mapas térmicos de una FPGA en funcionamiento, sino que los datos obtenidos permiten fijar claramente la posición del punto caliente. Se ha comprobado que efectivamente los consumos puntuales de potencia en el chip provocan gradientes de temperatura; una de las sorpresas es el valor de estos gradientes, que pueden llegar a los 5 °C.

La conclusión de esta primera tanda de experimentos es que existe una correlación clara entre consumo y gradiente de temperatura; esta relación es tan evidente que tiene sentido dar un paso adelante e intentar repetir estos experimentos sobre circuitos con una utilidad real, algo que se verá en el siguiente punto.

# 2. Experimentos sobre un circuito real

En este punto lo que se tratará de evaluar es si la información de los mapas térmicos también puede ser de utilidad en circuitos reales, muchísimo más grandes y con patrones de consumo mucho más complejos que los puntos calientes presentados en la sección anterior. Como caso de estudio se ha pensado utilizar un sistema con dos microprocesadores, de tal manera que el objetivo será comprobar si a través del mapa térmico se puede identificar cuál de ellos está consumiendo más.

El microprocesador que se ha escogido es Plasma [Rho01], un diseño VHDL disponible en *opencores.org* y que es prácticamente 100% compatible con MIPS-I. Las razones para esta elección son básicamente dos: primero por su plena utilidad real, y segundo, porque es hardware libre. No se trata de un limitado microcontrolador de 8 bits, sino que es un completo microprocesador de 32 bits, y además es compatible con una arquitectura muy empleada en aplicaciones reales. En efecto, MIPS, junto con ARM y PowerPC, son los microprocesadores más empleados en sistemas embebidos.

Para realizar los experimentos se han mapeado dos cores Plasma sobre una FPGA XCV800HQ240-4C. Utilizando directivas LOC, la posición del primero de ellos se ha restringido a las primeras 32 columnas de la FPGA a la izquierda, y la del otro, a las 32 primeras columnas de la derecha. Cada microprocesador tiene disponibles 4 KB de memoria, implementados en BlockRAM; por supuesto, el primer procesador utiliza la

BRAM de la izquierda, y el segundo, el de la derecha. Esta memoria esta iniciada con el programa que debe ejecutar cada CPU; para cambiar los programas no se ha reimplementado el diseño, sino que se han alterado los contenidos de la BlockRAM directamente sobre el archivo .bit (*bitstream*) con la herramienta DATA2BRAM [Xil02g].

Adicionalmente se han añadido 32 sensores, en forma de matriz de 4 x 8, para obtener el mapa térmico. El sensor utilizado no es el JBits, sino la macro descrita en la sección 8.2 del capítulo anterior. Esto es porque, como ya se indicó en aquel punto, no se consiguió que JBits no interfiriese en el rutado de los diseños ya presentes en el *bitstream*.

Para implementar este circuito se utilizó la herramienta ISE 5.1, con el sintetizador XST. En la siguiente tabla se pueden ver las estadísticas del circuito, y en la Fig. 119 se muestra el layout, donde se ve claramente la posición de los dos cores. Se ha remarcado la localización de los sensores de temperatura, que sólo ocupan un 5,7 % del área. La pobre velocidad máxima se debe a que el diseño original sólo está segmentado en dos etapas, lo que provoca una profundidad de lógica muy elevada, junto con un pobre diseño del banco de registros, que causa nodos con un fanout (y un retardo) muy elevado.

| | | |
|---|---|---|
| Características de área | IOBs | 20 (12%) |
| | Slices totales | 6240 (66%) |
| | Slices para sensores | 544 (5,7%) |
| | BlockRAMs | 16 (57%) |
| | Flip-flops | 2524 (13%) |
| | Tablas de lookup de 4 entradas | 8729 (46%) |
| Características de tiempo | Periodo mínimo | 82,387 ns |
| | Frecuencia máxima | 12,138 MHz |
| | Retardo medio de los 10 peores nodos | 13,239 ns |
| | Niveles de lógica | 29 |

Tabla 18: Estadísticas del circuito para mapas térmicos basado en dos microprocesadores

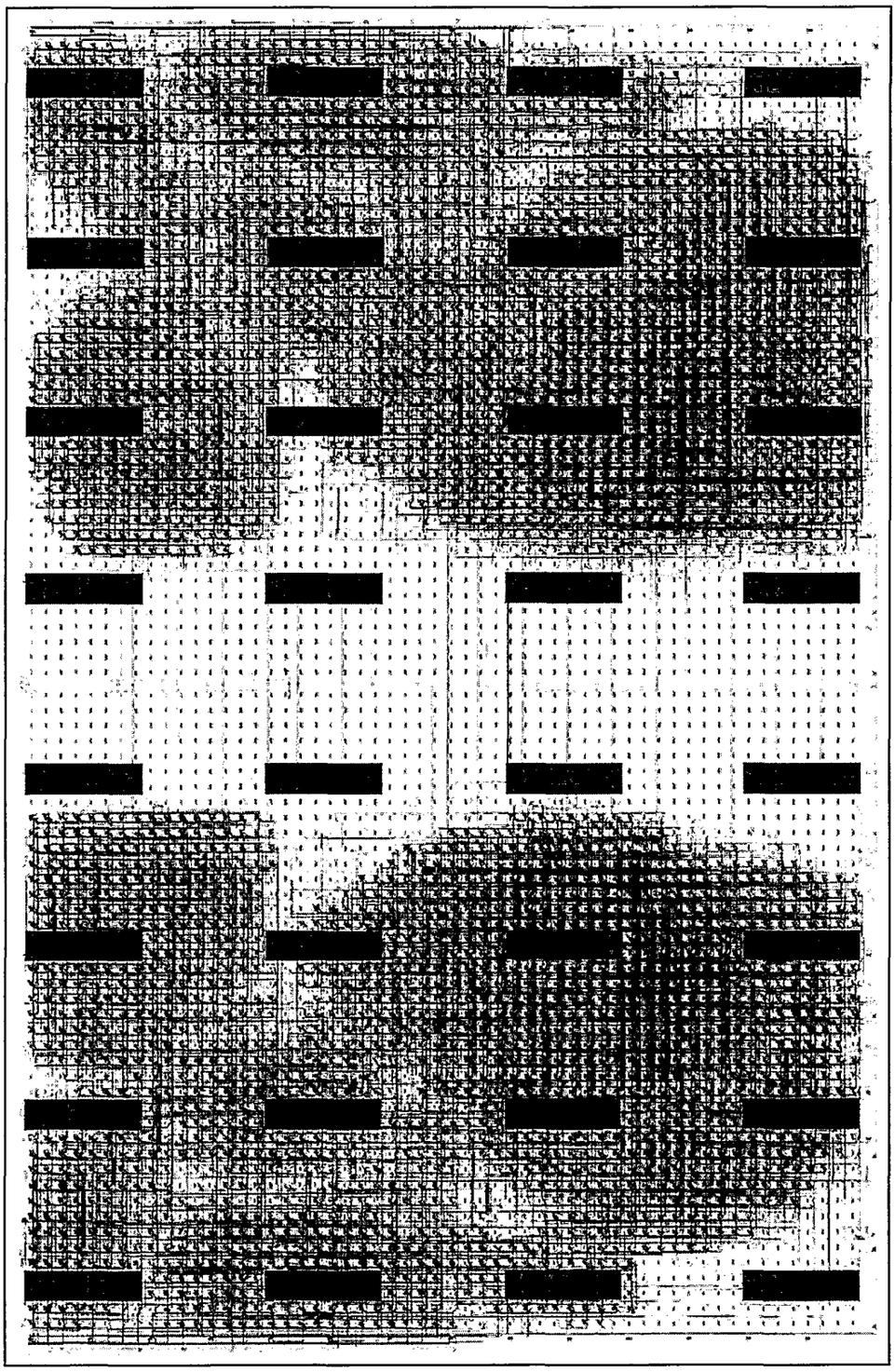


Fig. 119: Layout del circuito para mapas térmicos basado en dos microprocesadores

Se utilizaron dos programas de prueba diferentes. El primero, `opcodes.asm`, prueba todos las instrucciones del microprocesador y luego se queda colgado en un bucle infinito. El segundo, `pi.c`, calcula los primeros 40 dígitos del número $\pi$, usando aritmética de números enteros. Ambos venían en la distribución del propio core, que también incluye un ensamblador y un sencillo compilador de C, basado en gcc. La utilidad de `opcodes.asm` es comprobar que la CPU está funcionando correctamente; y además será el programa de bajo consumo (por el bucle infinito). Por otro lado, `pi.c` será el programa de alto consumo, al menos en comparación con `opcodes.asm`.

## 2.1. Ejecución de los experimentos

Los experimentos se llevaron a cabo sobre la placa AFX antes mencionada, a la que se conecto un analizador lógico para tener acceso a los puertos de depuración de los microprocesadores; de esta manera se podía saber si los programas se estaban ejecutando correctamente. Como frecuencia de reloj se escogió 10 MHz; los consumos a esta frecuencia quedan resumidos en la siguiente tabla:

| **Programa** | **$\mu$P izquierdo** | **$\mu$P derecho** |
|---|---|---|
| `opcodes.asm` | 92,75 mW | 93,75 mW |
| `pi.c` | 204 mW | 206,25 mW |

Tabla 19: Consumo de los diferente programas en cada microprocesador

Estos consumos se corresponden con el incremento de corriente que se produce al liberar el reset; con él activado todavía hay un pequeño consumo, de aproximadamente 25 mW para cada microprocesador.

En las siguientes figuras se representa el incremento de temperatura que se produce al liberar el reset de ambos microprocesadores. Para que las figuras sean más visibles, se representa la desviación con respecto al incremento medio; este incremento medio no aporta demasiado, salvo desplazar la escala, pues depende sólo de la potencia global, que ya se ha indicado en las tablas anteriores, y de la temperatura ambiente. Adicionalmente se han preparado termografías a partir de la interpolación 2D de las medidas de los sensores, que resultan todavía más intuitivas.

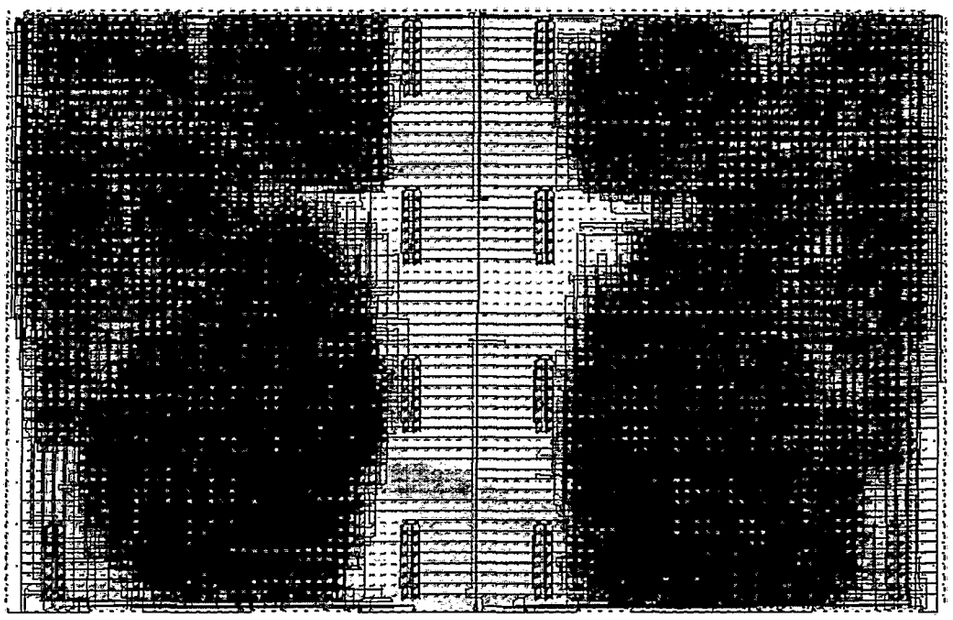


Fig. 120: Termografía obtenida mediante interpolación. Izqa. opcodes.asm, Dcha. pi.c

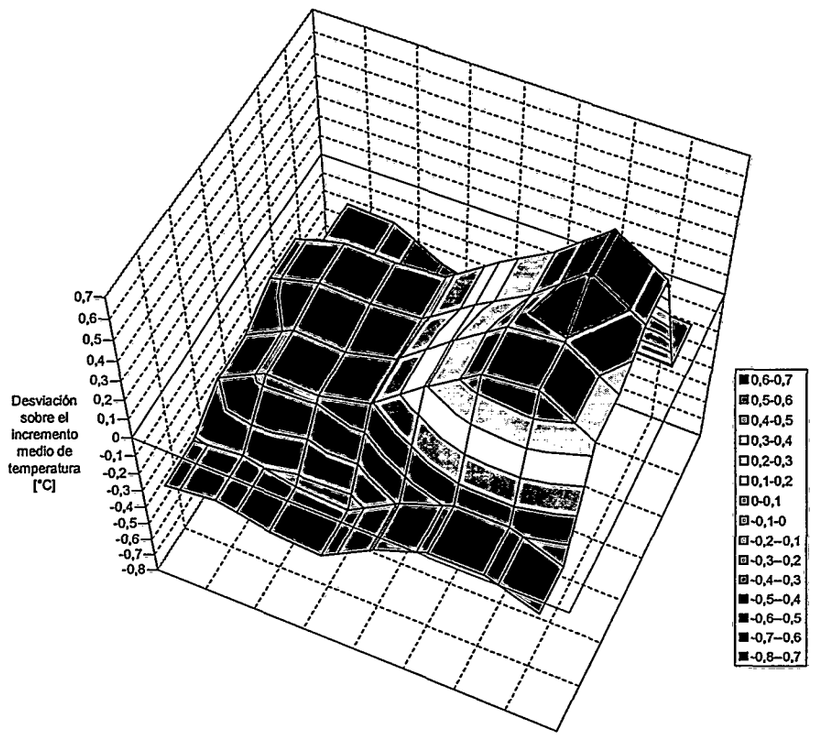



Fig. 121: Desviación sobre el incremento medio de temperatura. Izqa. `opcodes.asm`, Dcha. `pi.c`

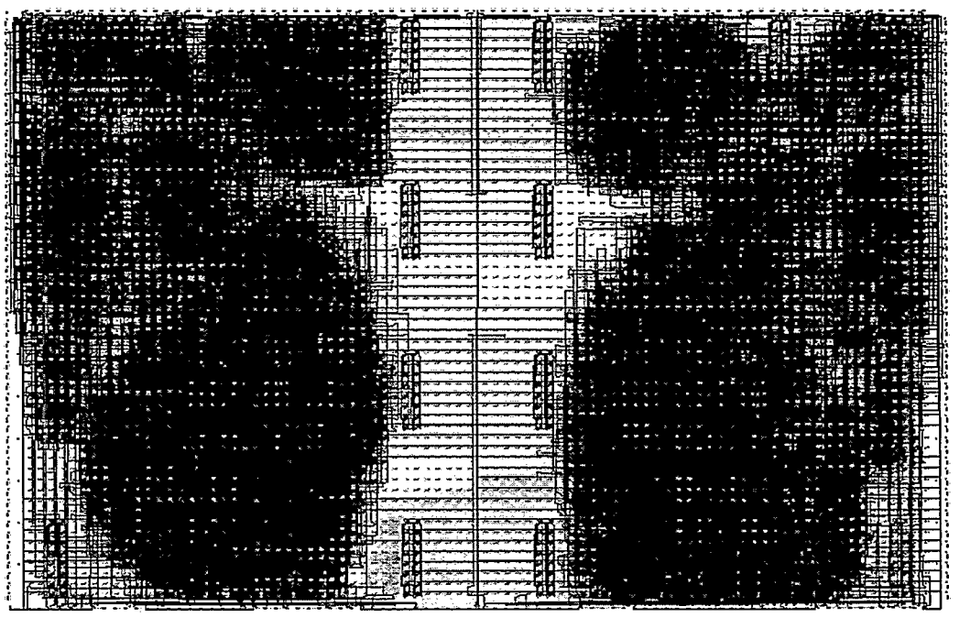


Fig. 122: Termografía obtenida mediante interpolación. Izqa. `pi.c`, Dcha. `opcodes.asm`

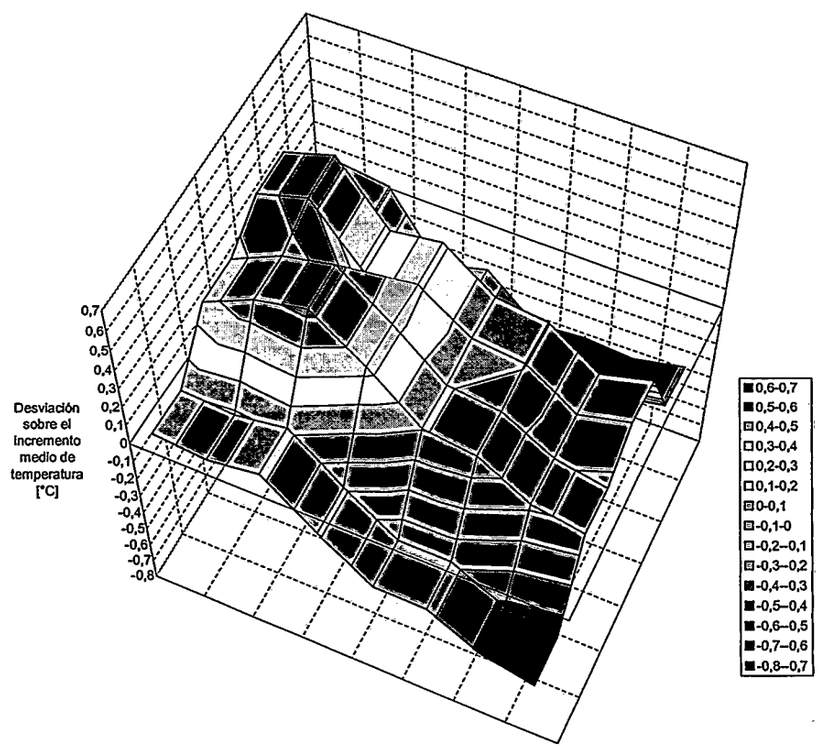



Fig. 123: Desviación sobre el incremento medio de temperatura. Izqa. `pi.c`, Dcha. `opcodes.asm`

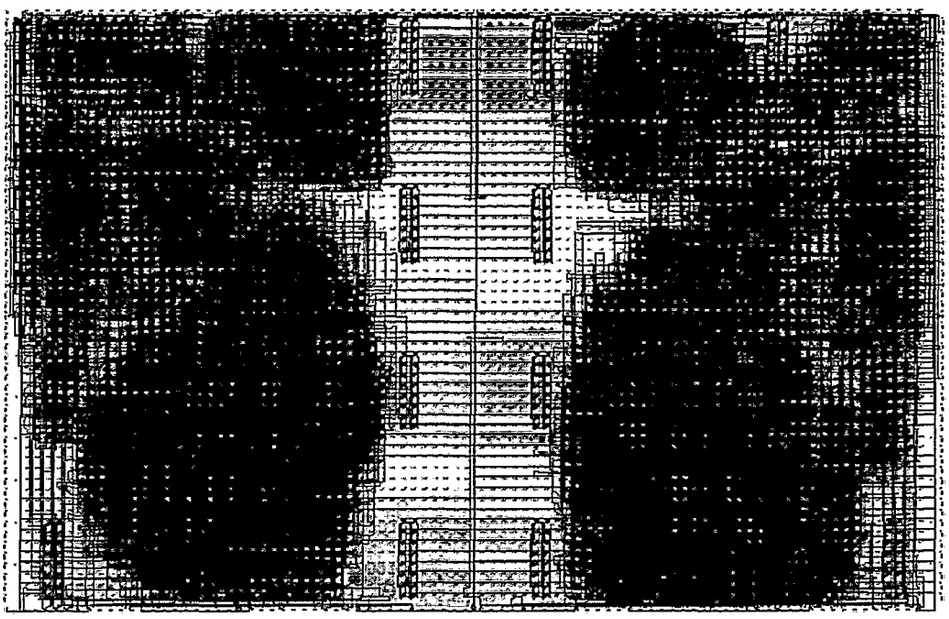


Fig. 124: Termografía obtenida mediante interpolación. A ambos lados `pi.c`

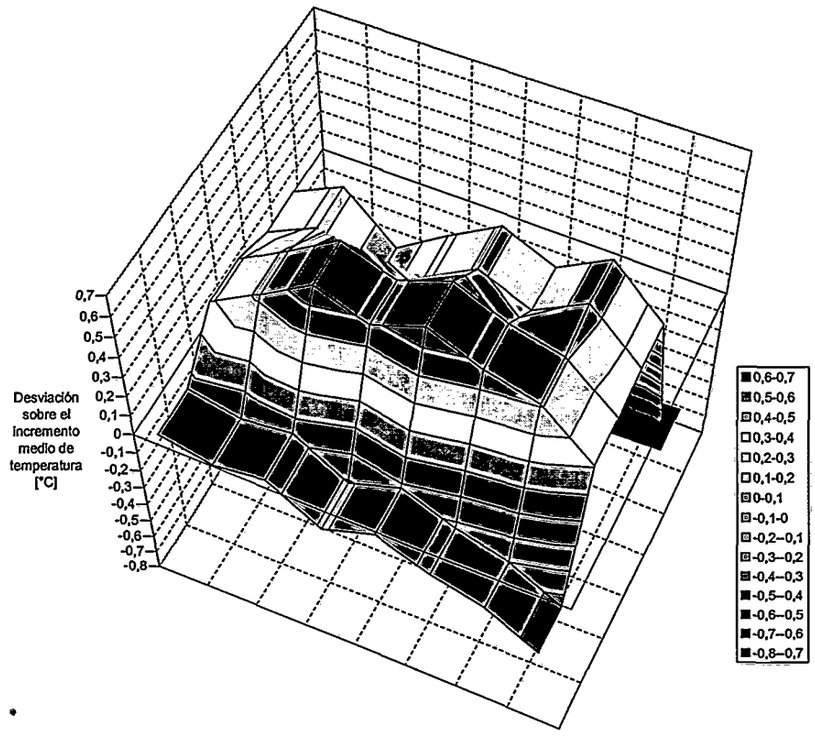



Fig. 125: Desviación sobre el incremento medio de temperatura. A ambos lados `pi.c`

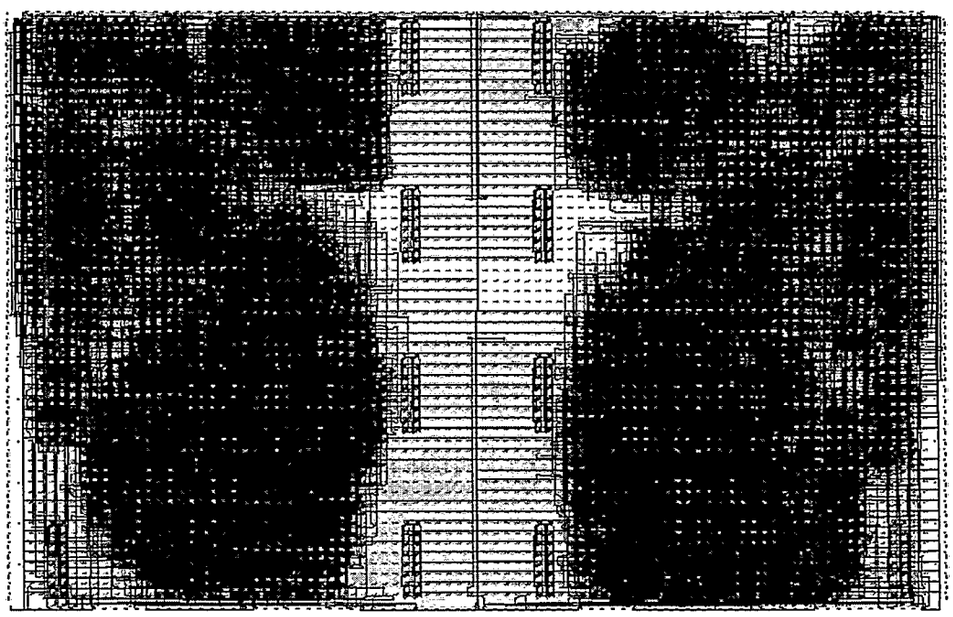


Fig. 126: Termografía obtenida mediante interpolación. A ambos lados `opcodes.asm`

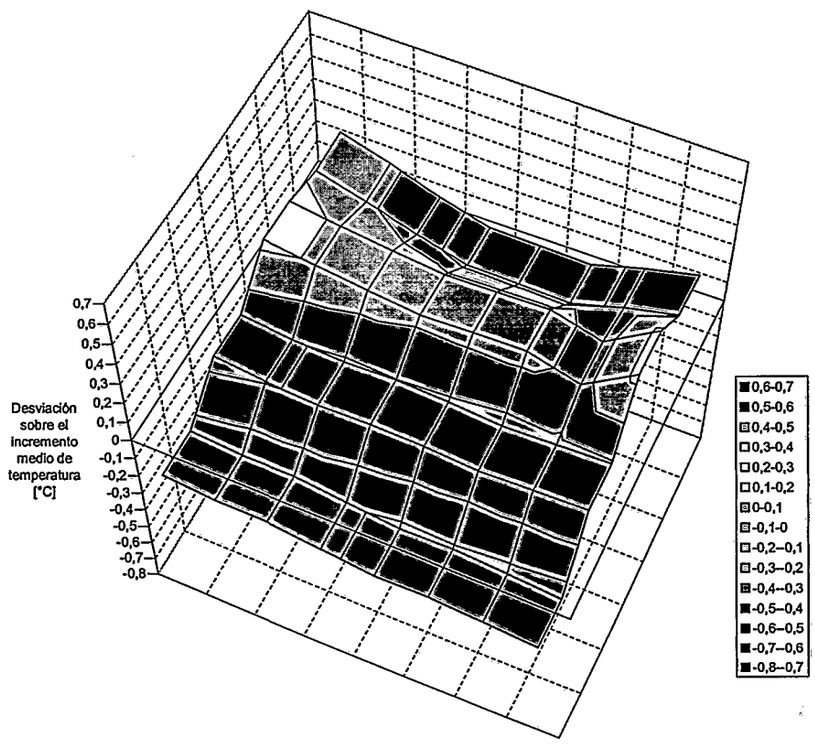



Fig. 127: Desviación sobre el incremento medio de temperatura. A ambos lados `opcodes.asm`

## 2.2. Resultados

Los resultados son positivos: se ve de manera muy clara que microprocesador es el que está disipando más potencia, y cuando ambos ejecutan el mismo programa, se produce una evidente simetría en los mapas térmicos. Además, se observa una diferencia notable en los microprocesadores entre la ejecución de un programa real y cuando están colgados. En este último caso los gradientes de temperatura se reducen *muy notablemente*.

Desgraciadamente, el proceso de emplazado y rutado entremezcla las localizaciones de las distintas unidades funcionales, por lo que sería demasiado aventurado tratar de averiguar que bloque de los microprocesadores es el que está consumiendo más, o a qué se deben las zonas más frías. El único caso en donde se puede ir un poco más lejos es en la Fig. 127, donde ambos microprocesadores están colgados, y sólo se observa una pequeña actividad en el acceso a las BlockRAMs, que están continuamente devolviendo `jump` y `nop`.

# 3. Conclusiones

Se ha demostrado que es posible realizar un mapa térmico de una FPGA en funcionamiento, y que además, los gradientes detectados informan verazmente de qué elementos del circuito están disipando más potencia. No sólo se han conseguido realizar estos experimentos en circuitos de prueba, sino que se ha logrado también obtener las termografías de un diseño tan real como un sistema con doble microprocesador, pudiendo observarse claramente las diferencias de consumo que provocan programas distintos.

Estos resultados abren la puerta a que se pueda conocer cuál es el consumo de los distintos bloques de un circuito implementado en una FPGA; esta técnica indirecta es con toda seguridad la única manera de lograr este objetivo. En el caso de que la FPGA se calentase por encima de los márgenes de seguridad, se actuaría sólo sobre los bloques culpables de ese incremento de temperatura, permitiendo que el resto de los elementos siguiesen funcionando normalmente. No tiene sentido aplicar una estrategia global de reducción de consumo (p.ej., bajar la frecuencia de reloj) cuando puede ser que sólo unos pocos bloques sean los que están consumiendo la mayor parte de la potencia.

En la actualidad, el tamaño que tienen las FPGAs permite implementar todo un sistema en un chip, y este crecimiento no ha parado. Los diseños que se implementan en FPGAs tan enormes son heterogéneos, incluyendo normalmente elementos disjuntos: uno o varios microprocesadores, rutas de datos para el procesado de señal, periféricos,... Esto se traduce en un consumo de potencia también heterogéneo, que justifica plenamente el uso del mapeado térmico.

Por último, hay errores que provocan un elevado consumo puntual de potencia. Por esta razón, estos errores suelen ser desastrosos, pues el incremento de temperatura y/o corriente termina provocando un fallo en el chip. Como se ha podido ver en todos los experimentos, el mapeado térmico sirve para detectar incrementos puntuales de consumo y fijar su posición, de tal manera que se pueda actuar sobre el error sin parar el resto del circuito.

# Capítulo 7.

# Conclusiones y trabajos futuros

## 1. Principales aportes de esta tesis

En esta tesis se han propuesto y validado una serie de ideas para implementar una estrategia de supervisión térmica de sistemas reconfigurables. La idea central es sencilla: una FPGA se puede configurar durante unos breves instantes con un oscilador en anillo y un contador, de manera de capturar la temperatura del silicio. La utilización extensiva de esta técnica en combinación con la capacidad de reconfiguración dinámica de las FPGAs permite construir mapas térmicos en tiempo real de los dispositivos.

La solución propuesta, sencilla y económica, es completamente novedosa en el campo de las FPGAs y había sido pasada por alto por los principales fabricantes de estos dispositivos. También es potencialmente importante para el estudio de las características térmicas de los encapsulados, al ofrecer claras ventajas sobre las otras técnicas disponibles hasta el momento.

Las principales aportaciones de esta tesis se pueden resumir en los siguientes puntos:

**1.** Se ha demostrado la utilidad de los osciladores en anillo como sensores de temperatura para FPGAs. Se han caracterizado diversas opciones que combinan diferentes relaciones entre retardos de LUTs y retardos de pistas. En todos los casos, se ha obtenido una respuesta lineal de frecuencia en función de la temperatura. Los sensores basados en CLBs pueden situarse virtualmente en cualquier posición de la pastilla y presentan una sensibilidad relativa muy similar, independientemente de la

posición sobre el dado de silicio. Los sensores basados en un IOB, tienen mínima ocupación y pueden usarse en aquellos casos en que los recursos internos de las FPGAs estén completamente ocupados.

**2.** Complementariamente al punto anterior, se ha verificado experimentalmente la utilidad como sensores de temperatura de los diodos de enclavamiento ubicados en las patas de la FPGAs. Esta idea había sido propuesta por Peter Alfke, en un correo electrónico enviado a grupo `comp.arch.fpga`, pero no se habían publicado datos concretos de sensibilidad, rango, y variación con Vcc. Como principal resultado, se ha verificado que los diodos de enclavamiento pueden ser útiles como transductores de temperatura debido a su mínima sensibilidad frente a variaciones en la tensión alimentación.

**3.** Se han desarrollado dos procedimientos de calibración. Uno de alta precisión, que requiere un horno de temperatura controlada, y otro menos exacto pero simple y económico, que permite montar rápidamente una monitorización térmica con un error cercano al 5%, sólo con normalizar la respuesta de los sensores.

**4.** Esta Tesis espera abrir un nuevo campo de aplicación para las FPGAs. La capacidad del reconfiguración transforma a estos dispositivos en una herramienta poderosa para el estudio de los aspectos térmicos circuitos integrados y encapsulados. Permite a los investigadores del tema realizar un número ilimitado de nuevos experimentos. Simplemente la posibilidad de "mover" el sensor a lo largo del *chip* es impensable en cualquier otra tecnología. Quizás por estas características, algunos resultados de esta investigación se publicaron rápidamente en el *IEEE Transactions on Components and Packaging Technologies (CPM),* una revista totalmente alejada de la temática de las FPGAs y los sistemas reconfigurables.

**5.** Desde una perspectiva práctica, la metodología propuesta puede ayudar a la construcción de arquitecturas muy fiables basada en FPGAs, como las requeridas en aviónica y electrónica espacial. La técnica permite detectar y eliminar puntos calientes dentro del dado de silicio, una posibilidad que ninguna otra opción tecnológica hasta el momento era capaz de realizar en condiciones normales de operación (*chip* con encapsulado, disipador, dentro del chasis, etc.). En efecto, aunque las ideas de limitación térmica (en inglés *deration*), consistentes en bajar la temperatura de operación para conseguir un margen extra de fiabilidad, están siendo cuestionadas [Lal96, Jac97,

Pec97], lo que en la actualidad parece indiscutible es que los gradientes intensos y las variaciones pronunciadas en cortos periodos de tiempo (*thermal shocks*) son muy negativos para la vida de un circuito integrado. Estos dos fenómenos se pueden detectar perfectamente con la técnica propuesta en esta Tesis.

**6.** Los osciladores en anillos pueden utilizarse simplementa para medir retardos de bloques combinacionales, tal como se explica en la Patente de B. Conn que se comenta más adelante. Esta idea se desarrolló independientemente en esta tesis y se utilizó, dentro del ámbito industrial local, para la caracterización de las LUTs del FPSOC, la primera FPGA española fabricada por SIDSA, una compañía *fabless* de semiconductores que mantiene fluidas comunicaciones con la UAM. En particular, la idea fue utilizada por el Dr. Julio Faura, anterior doctorando del director de esta tesis, para determinar retardos en las primeras muestras del *chip*.

**7.** Una punto interesante de las aportaciones de esta tesis al mundo de la ingeniería, es el impacto que sus resultados hayan podido tener sobre la propia Xilinx, la fabricante de las FPGAs. En 1998, con la aparición de la familia Virtex, esta compañía optó por incluir un diodo embebido en el chip para sensar la temperatura. Esta decisión probablemente se tomó siguiendo el camino adelantado por Intel, que desde unos años antes ofrecía esta posibilidad en su microprocesador Pentium. La utilización de esta técnica en una CPU de tanto éxito comercial propició la aparición de multitud de soluciones que integraban en un único chip todos los circuitos necesarios para medir la temperatura en el diodo; un ejemplo fue presentado en el capítulo 2.

En esta tesis no sólo se demostró una utilidad de los CLBs para medir la temperatura, sino que además se caracterizó la célula OSC4, embebida en la serie XC4000 de Xilinx. Se demostró su gran valor como sensor de temperatura, pues exhibió los mejores resultados en rango de frecuencia, ocupación y sensibilidad frente a variaciones en Vcc. La utilización de la célula de Xilinx OSC4 como sensor de temperatura no había sido imaginada por sus propios creadores. Más bien lo contrario: en el manual de componentes se indicaba que se debía tener precaución con el bloque OSC4 debido a su variación con la temperatura, tal como se detallaba en la sección 1.3 del capítulo 4.

Durante el mes de junio de 1998, el director de esta tesis mantuvo un breve encuentro en California con Peter Alfke, el Director de Aplicaciones de Xilinx, célebre por sus magníficas notas de aplicación sobre diseño con FPGAs, publicadas a lo largo de

muchos años en la parte final de los manuales de Xilinx. A este contacto siguieron una serie de correos electrónicos donde se discutieron los principales resultados preliminares (1998) de esta tesis [Alf98, Alf98a, Alf99, Alf99a]. Éstos fueron posteriormente difundidos dentro de Xilinx mediante un correo interno, cuya copia se envió al director de esta Tesis y que se reproduce en el cuadro adjunto.

*I just got hold of an excellent paper "Thermal Testing on Programmable Logic Devices" (Proceedings IEEE ISCAS 1998 Vol. II, pages 240-243 ) by three Spanish academics from the Universidad Autonoma de Madrid.*

*Here are their conclusions. I will try to get permission to reprint the whole paper as a Xilinx app. note.*

*XC4000 die temperature can be measured accurately with on-chip ring oscillators, each built in two CLBs by concatenating LUTs with one inversion around the ring. Depending on the net- and LUT-delays, the frequency of oscillation was between 10 and 16 MHz, decreasing linearly with temperature at a rate of minus 0.2% per degree C. (They measured three different LUT/net ratios, with 12.6/26.0, 12,6/11.6, and 23.2/11.8 of LUT/net delays in ns. The frequency coefficients were 0.197, 0.198, and 0.196% per degree C. I call that convincing !)*

*Our own on-chip oscillator was measured at a higher value of 0.25% per degree C, but was instead very stable over voltage, changing only +0.14% in frequency for every % of voltage change. The LUT-based oscillator frequencies changed 0.6% per % voltage change ( 0.595, 0.632, and 0.591% to be exact. )*

*They also measured the forward-biased voltage drop of the I/O protection diode, and found it to be minus 0.13 mV per degree C, but not as linear as the frequency change.*

*This work was done completely independent of the work at Xilinx ( by Bob Conn et.al.), and the results are comparable. Good stuff !*

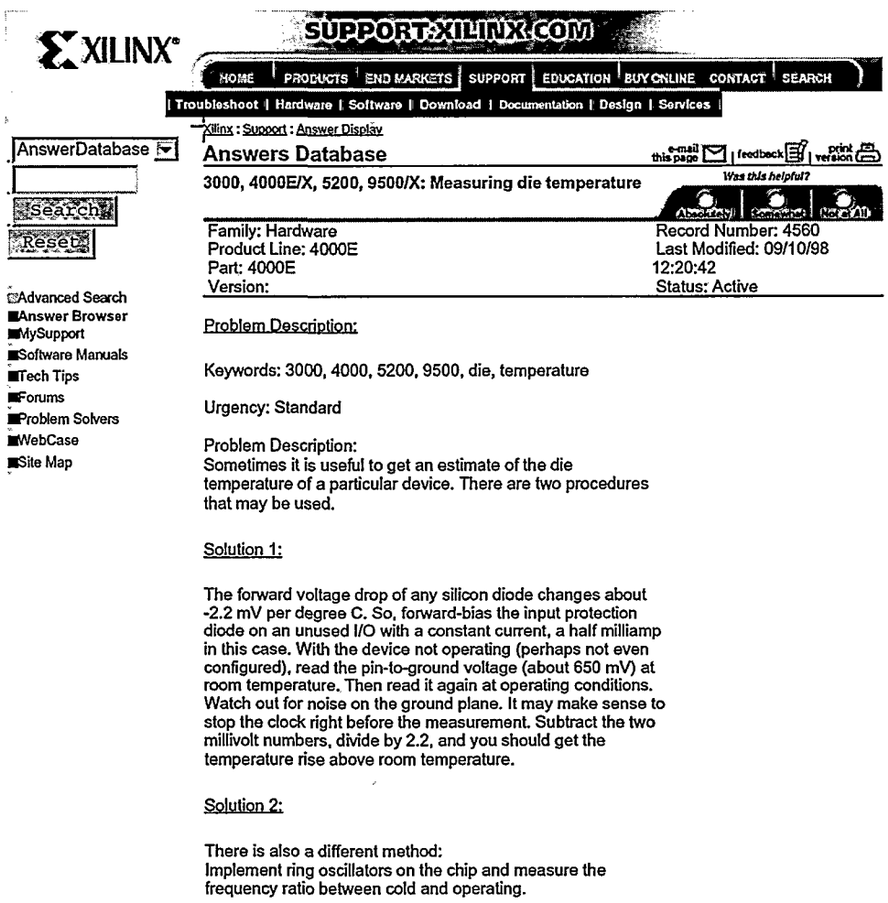
*While it is more difficult to simulate and predict the power consumption and junction temperature of programmable devices, their programmability offers ways to measure the die temperature digitally, very accurately, and almost for free.*

*Hurray for the experimenter.*

*Peter*

De este correo se deduce que Xilinx tenía datos similares sobre retardos en función de la temperatura, pero que no los había asociado con la posibilidad de medir temperatura. La compañía conocía la utilidad de los osciladores en anillo como método para medir retardos. Lo demuestra la patente *"US5790479: Method for characterizing interconnect timing characteristics using reference ring oscillator circuit"*, a favor de Robert Conn, presentada el 17 de septiembre de 1996 y otorgada en agosto de 1998.

Sin embargo, en los meses siguientes la idea, tal como se plantea en esta Tesis, ya apareció en la base de datos de soporte para clientes que Xilinx mantiene en Internet, en el *Answer Record No. 4560*, reproducida en la siguiente figura. De este modo, la idea se difundió por "canal oficial" a la comunidad de usuarios de FPGAs.



XILINX®

SUPPORT.XILINX.COM

HOME | PRODUCTS | END MARKETS | SUPPORT | EDUCATION | BUY ONLINE | CONTACT | SEARCH

Troubleshoot | Hardware | Software | Download | Documentation | Design | Services

Xilinx : Support : Answer Display

AnswerDatabase

Search

Reset

Advanced Search
Answer Browser
MySupport
Software Manuals
Tech Tips
Forums
Problem Solvers
WebCase
Site Map

**Answers Database**

e-mail this page | feedback | print version

Was this helpful?

**3000, 4000E/X, 5200, 9500/X: Measuring die temperature**

| | |
|---|---|
| Family: Hardware | Record Number: 4560 |
| Product Line: 4000E | Last Modified: 09/10/98 |
| Part: 4000E | 12:20:42 |
| Version: | Status: Active |

Problem Description:

Keywords: 3000, 4000, 5200, 9500, die, temperature

Urgency: Standard

Problem Description:
Sometimes it is useful to get an estimate of the die temperature of a particular device. There are two procedures that may be used.

Solution 1:

The forward voltage drop of any silicon diode changes about -2.2 mV per degree C. So, forward-bias the input protection diode on an unused I/O with a constant current, a half milliamp in this case. With the device not operating (perhaps not even configured), read the pin-to-ground voltage (about 650 mV) at room temperature. Then read it again at operating conditions. Watch out for noise on the ground plane. It may make sense to stop the clock right before the measurement. Subtract the two millivolt numbers, divide by 2.2, and you should get the temperature rise above room temperature.

Solution 2:

There is also a different method:
Implement ring oscillators on the chip and measure the frequency ratio between cold and operating.

Fig. 128: Extracto del *Answer Record No. 4560*

**8.** Una conclusión de esta tesis no estrictamente relacionada con asuntos térmicos es que JBits ha demostrado ser una herramienta muy potente a la hora de crear diseños reconfigurables en tiempo de ejecución. Sin entrar en detalles acerca de las ventajas e inconvenientes de esta herramienta, que se discuten mejor en el apéndice A, los resultados en general han sido muy positivos. Sólo ha habido un punto mejorable: la interfaz de JBits con circuitos diseñados usando las herramientas convencionales.

**9.** Una de las mayores críticas a los osciladores en anillo es su fuerte sensibilidad con respecto a las variaciones en la tensión de alimentación. En esta tesis se ha desarrollado una metodología que permite compensar esta dependencia. Y no sólo eso, sino además obtener simultáneamente los valores de la temperatura y la tensión de alimentación. Sólo hay que construir dos sensores con respuestas diferentes (algo tan sencillo como variar el rutado del oscilador en anillo), para hacerlo más largo o más corto.

**10.** Probablemente el resultado más novedoso de esta tesis sea la posibilidad de construir mapas térmicos de FPGAs en funcionamiento, sin tener que parar la actividad del circuito implementado en ella, y sin utilizar ninguna pata de E/S, dado que todo el proceso se realiza a través del puerto de configuración.

Hasta ahora, la inmensa mayoría de las termografías presentadas en los artículos de investigación se habían obtenido usando cámaras infrarrojas. Este método, que en un principio puede parecer muy potente, es completamente inútil para circuitos comerciales, pues necesita de un acceso visual directo al silicio. No sólo hay que hacer una ventana en el encapsulado, sino que también hay que eliminar los disipadores que pudiera tener adosados. El resultado es una configuración totalmente alejada de la realidad.

En esta tesis se ha demostrado que es posible crear estas termografías empleando una matriz de sensores. Y no sólo eso, también que han resultado ser una herramienta muy potente para conocer que zonas del circuito están disipando más potencia. Se ha probado su utilidad tanto con puntos calientes diseñados *ad-hoc*, como en circuitos reales como el MIPS-I. En particular, se han presentado figuras que muestran muy claramente como se pueden emplear estos mapas térmicos para determinar el consumo en cada parte de un sistema biprocesador.

Por otro lado, esta aplicación también ha servido para demostrar la potencia de la reconfiguración parcial en la familia Virtex de Xilinx: se pueden reconfigurar un bit de un

*frame* sin que se produzcan *glitches* en el resto de bits de configuración, o sea, sin que sea necesario parar el resto del circuito. Aunque esta afirmación estaba claramente expresada en las notas de aplicación de Xilinx, pocos diseñadores eran conscientes de ello, e incluso algunos de ellos siguen manteniendo que siempre que se reconfigure un *frame*, hay que parar los circuitos cuya configuración utilice este *frame*.

# 2. Publicaciones

Las principales aportaciones de esta tesis han sido publicadas en una serie de artículos en revistas y congresos sobre FPGAs y diseño electrónico. Cabe destacar que el primer artículo sobre el tema fue publicado en el FPL de 1997, sobre experimentos preliminares realizados el año anterior. Los dos artículos más importantes que resumen la tesis son:

- S. Lopez-Buedo, J. Garrido y E. Boemo, "Dynamically Inserting, Operating, and Eliminating Thermal Sensors of FPGA-based Systems", *IEEE Transactions on Components and Packaging Technologies (CPM)*, Vol. 25, No. 4, pp. 561-566, diciembre de 2002.
- S. Lopez-Buedo, J. Garrido y E. Boemo, "Thermal Testing on Reconfigurable Computers", *IEEE Design & Test of Computers*, pp. 84-90, enero-marzo de 2000.

Además de los artículos anteriores, otros resultados parciales fueron publicados en:

- S. Lopez-Buedo, P. Pernas, P. Riviere y E. Boemo, "Run-time Reconfiguration to Check Temperature in Custom Computers: An Application of JBits Technology", *Lecture Notes in Computer Science*, Vol. 2438, pp. 162-170, Berlín: Springer-Verlag, 2002.
- S. Lopez-Buedo, J. Garrido y E. Boemo, "Measurement of FPGA Die Temperature Using Run-time Reconfiguration", *Proc. Therminic 2001 (Int. Workshop on Thermal Investigations of ICs and Systems)*, pp. 168-173. París, septiembre de 2001. TIMA-CMP 2001.
- S. Lopez-Buedo, J. Garrido y E. Boemo, "Thermal Testing on Programmable Logic Devices", *Proc. 1998 IEEE ISCAS (Int. Symp. on Circuits and Systems)*, Vol. II, pp.240-243, Monterrey, California, junio de 1998.

- S. Lopez-Buedo y E. Boemo, "A Method for Temperature Measurement on Reconfigurable Systems", *Proc. XII DCIS Conference (Design of Circuit and Integrated Systems)*, pp. 727-730, Universidad de Sevilla: noviembre de 1997.
- E. Boemo y S. Lopez-Buedo, "Thermal Monitoring on FPGAs using Ring-Oscillators", *Lecture Notes in Computer Science*, No. 1304, pp. 69-78, Berlín: Springer-Verlag, 1997.

# 3. Trabajos futuros

Esta tesis espera constituirse en el inicio de una nueva línea de investigación. Los resultados que se ha presentado son muy prometedores, y todo parece indicar que con esta técnica se pueden conseguir aplicaciones muy novedosas. Esta sección trata de identificar los principales caminos a seguir. Cada uno de ellos podría constituirse en tema de una nueva tesis.

**1. Generalización de los resultados.** El primer punto debe ser dar universalidad a las técnicas de medida de temperatura presentadas en esta tesis. En particular, se debería avanzar sobre los siguientes puntos:

- Las medidas de sensores de temperatura se deberían extender a distintas muestras de un dispositivo y a todos los dispositivos de una familia. Con estos resultados se podría hacer un estudio de la dispersión en las características de estos sensores, que permitiese hacer una estimación rigurosa del error cometido en las medidas. Una investigación de este tipo, por su coste, sólo se podría realizar bajo contrato con Xilinx.
- En la misma dirección, debería también estudiarse si el envejecimiento del circuito afecta a la respuesta de los sensores de temperatura.
- Finalmente, no se deberían limitar a un único fabricante de FPGAs. Estas técnicas deberían validarse como mínimo en dispositivos de Altera (directa competencia de Xilinx) y Actel (por su tecnología alternativa).

**2. Desarrollo de una metodología para conocer el consumo detallado de cada módulo implementado en la FPGA.** En esta tesis se ha demostrado que es posible crear mapas térmicos de FPGAs en funcionamiento, y que los gradientes detectados informan fielmente sobre las áreas del chip en las que se está consumiendo más

potencia. Todo esto abre la puerta a la posibilidad de que se pueda conocer el consumo por separado de cada uno de los módulos que se han implementado en el dispositivo. Además, conecta muy bien con la tendencia actual de migrar al mundo de las FPGAs el estilo de diseño basado en bloques, típico de los ASICs: ejemplos de esto son el *modular design* de Xilinx o el *LogicLock* de Altera. Pero antes de que esto sea posible, quedan los siguientes puntos por resolver:

- El primero de ellos, y probablemente el más sencillo, es la calibración. Ya se ha demostrado que sensores con el mismo layout se comportan prácticamente igual, incluso si se cambia de dispositivo. Por eso no es necesario calibrarlos todos, basta con medir sólo un sensor y usar normalización para el resto. Aunque el error cometido con esta estrategia es muy pequeño, menos de ±1 °C, los mapas térmicos deben hacerse con mucha más precisión. Debe buscarse algún método alternativo de calibración, que permita compensar estas diferencias, y que no sea tan complicado como la medida en un horno de temperatura controlada,. Una posibilidad podría ser emplear un *bitstream* que calentase uniformemente la FPGA, sin gradientes de temperatura, y la respuesta de los sensores se ajustaría hasta que todos midiesen el mismo valor.

- Un problema mucho más complejo es tener un modelo térmico preciso de la FPGA. Aquí las dificultades provienen de la cantidad de parámetros desconocidos que hay: tamaño del chip, detalles tecnológicos tanto del silicio como del encapsulado... Además, la gran mayoría de estos datos son secreto industrial de los fabricantes. Debería desarrollarse una metodología que permitiese realizar un modelado bajo incertidumbre, partiendo de los gradientes obtenidos con puntos calientes de consumo conocido. Un buen punto de partida podrían ser los resultados presentados en la primera parte del capítulo 6.

- Por supuesto, otro problema importante es la estimación de consumo. Se deben desarrollar herramientas que permitan desglosar el consumo por elementos de la FPGA, y que sean precisas no tanto en términos absolutos, sino más bien de una manera relativa: cuantas veces más consume un bloque en relación con otro. Se debería estudiar si las herramientas proporcionadas

por los fabricantes, como Xpower, pueden ser de utilidad, y en caso contrario, desarrollar herramientas propias.

- Finalmente, un problema menor: si ya se demostró en el capítulo 2 que a partir del consumo no se puede predecir de manera fiable la temperatura del silicio, el inverso también es cierto. Si se quiere conocer el consumo absoluto de cada bloque implementado en la FPGA, se debe medir el consumo global del chip. Si no, sólo se podrán tener medidas relativas (que en cualquier caso son muy interesantes también).

**3. Extensión hacia la medida dinámica de la temperatura.** En esta tesis, prácticamente todas las medidas de temperatura que se han presentado son estáticas, una vez alcanzado el equilibrio térmico. Sin embargo, las técnicas de medida presentadas permiten también hacer medidas dinámicas; en particular, los osciladores en anillo pueden medir la temperatura muy rápidamente, con anchos de banda de KHz a MHz. Una de las consecuencias de esta extensión es que habría que ampliar también el modelo térmico comentado en el punto anterior, de tal manera que permitiese predecir la evolución temporal de la temperatura en la FPGA al activarse un punto caliente.

**4. Desarrollo de una metodología para la detección de fallos en la FPGA.** Para este punto, serían necesario desarrollar antes los dos anteriores. Con medidas estáticas sólo se pueden detectar los fallos que originan evidentes puntos calientes en la FPGA. Esto es suficiente para los errores más graves, aquellos que pueden poner en peligro la vida del dispositivo. Pero fallos más pequeños pueden pasar desapercibidos, enmascarados entre el consumo del resto de elementos del circuito. Sin embargo, la combinación de medidas estáticas con medidas dinámicas puede ser una herramienta muy potente a la hora de detectar fallos, siguiendo la línea de los grupos punteros en el tema [Alt01].

- Cuando un fallo se activa genera una "onda de calor" que se propaga por todo el chip; un buen ejemplo de esto son las figuras presentadas en la sección 5 del capítulo 4. Una alternativa para saber la localización del fallo es medir cuanto tiempo tarda en llegar esta onda de calor a por lo menos tres sensores, utilizando triangulación.

- Siguiendo con la utilidad de las medidas dinámicas, si por ejemplo sólo se detecta el incremento de calor cuando un microprocesador accede a un periférico, esto indicaría un problema en su interfaz, como una colisión de buses.

**5. Utilización de estas técnicas para caracterizar encapsulados.** Como se vio en el capítulo 2, los encapsulados se caracterizan empleando dados de prueba. La justificación para esta metodología vienen de que los chips normalmente no tienen ni sensores de temperatura ni elementos que permitan calentarlos uniformemente, como las resistencias difundidas de estos chips de prueba.

- En esta tesis ya se ha visto que en FPGAs esto no es así, porque se puede disponer de prácticamente tantos sensores de temperatura como se quiera, en cualquier posición del dado. Y además, existe una solución muy sencilla para calentar el chip: no hay más que poner flip-flops y tablas de *lookup* a conmutar.
- Utilizar el dispositivo final en lugar de un dado de prueba es una gran ventaja; parámetros tan básicos como la resistencia térmica dependen del propio chip, por lo que emplear dados de prueba no deja de ser algo un tanto engañoso.
- Además, empleando medidas dinámicas se podrían conocer otros datos muy interesantes, como por ejemplo el espectro de constantes de tiempo térmicas. Este dato, que normalmente se obtiene empleando dados de prueba especiales y un complejo equipamiento [Mic01], ofrece una información muy valiosa acerca de la calidad de las uniones térmicas del chip al encapsulado, y de éste al disipador.

**6. Uso de esta técnica como herramienta de investigación.** Complementariamente al punto anterior, estas técnicas son una poderosísima herramienta de investigación en el campo tanto del diseño de encapsulados como en la gestión térmica de los mismos, porque los datos se obtienen del propio dispositivo final, una FPGA real.

- Los fabricantes de encapsulados podrían probar nuevos diseños con la seguridad de que los resultados son directamente utilizables en la práctica, porque las medidas se hacen sobre dispositivos reales.
- De la misma manera, las FPGAs se podrían convertir en una excelente plataforma de bajo coste para estudiar y desarrollar nuevos tipos de disipadores.

# Apéndice A.

# Introducción a JBits

El objetivo de este apéndice es hacer una descripción práctica de JBits, que pueda servir como punto de partida para empezar a trabajar con él. JBits es una API (*application program interface*) y un conjunto de aplicaciones, que están escritos en Java, y que permiten crear, modificar o depurar diseños para FPGAs de Xilinx; la principal característica de JBits es que está especialmente orientado a la reconfiguración en tiempo de ejecución.

## 1. ¿Qué es JBits?

Como toda API de Java, JBits está compuesto por una colección de clases y métodos. Pero sería un tanto injusto considerar a JBits sólo como un API; con él vienen toda una colección de herramientas de ayuda, que usan en mayor o menor medida este API.

La API de JBits nos ofrece por un lado la posibilidad de controlar a muy bajo nivel los bitstreams de configuración. Permite leerlos o modificarlos a nivel de bit, algo inédito para un dispositivo comercial (salvando la infortunada familia 6200 [Xil97]).

Por otro lado, JBits permite realizar circuitos empleando una metodología estructural, a base de cores parametrizables, usando los llamados RTP cores (*run-time parameterizable*, parametrizables en tiempo de ejecución). Esta es sin duda la manera más adecuada de diseñar, pues permite crear circuitos de una manera rápida, a base de bloques con una funcionalidad de medio-alto nivel (desde sumadores, contadores,... hasta CORDIC, filtro FIR,...)

El apelativo de RTP core, o sea, parametrizable en tiempo de ejecución, viene del hecho de que los cores están construidos como clases de Java que contienen el core prediseñado. Cuando se instancia un nuevo core, su clase correspondiente es capaz de generar inmediatamente el bitstream, sin tener que pasar por las herramientas convencionales de emplazado y rutado. Es por esta razón que JBits se puede emplear para diseños que cambien dinámicamente, en tiempo de ejecución.

## 1.1. Aportaciones de JBits

Siguiendo con este tema, probablemente la mejor aportación de JBits es que encapsula de una manera muy efectiva todos los detalles de funcionamiento de la reconfiguración parcial en los dispositivos de la familia Virtex de Xilinx. Dados dos bitstreams, JBits es capaz de inferir dónde están las diferencias y reconfigurar sólo la parte de la FPGA que cambia, además sin alterar el normal funcionamiento del resto del circuito. Y empleando sólo unas pocas líneas de código, como se verá más adelante.

Se puede establecer un paralelismo entre lo que representa Java con respecto a la programación estándar y lo que significa JBits con respecto al diseño hardware convencional. Tanto en el diseño HW como en la programación convencionales se hace un esfuerzo muy grande durante en la etapa de compilación para obtener un producto completamente optimizado para la plataforma en la que se va a ejecutar: bien se obtiene un código máquina para un microprocesador en concreto, o un bitstream para un modelo en particular de FPGA. Sin embargo, en el modelo Java/JBits el objetivo es otro, obtendremos un producto genérico, del cual no conocemos dónde va a ser ejecutado. Parte del esfuerzo se lleva al tiempo de ejecución, bien en forma de interpretar el código Java, o bien en forma de generar el bitstream para la FPGA en particular que se esté usando.

JBits ofrece una también la ventaja para el codiseño hardware-software que puede ser utilizado para, usando sólo Java, crear un producto mixto que junte hardware y software, y que además sea multiplataforma, y que se pueda configurar en tiempo de ejecución. Utilizando el modelo convencional de desarrollo, el codiseño hardware-software queda limitado a algo estático, donde las decisiones se toman en tiempo de diseño, y no pueden ser dinámicamente variadas. Sin embargo, con JBits se puede decidir dinámicamente que partes se ejecutan en HW y cual en SW, y además todo estar empaquetado en Java

## 1.2. Inconvenientes de JBits

En el otro lado de la balanza, la mayor crítica que se le puede hacer a JBits es su carencia de un estilo de diseño comportamental, pues el diseño estructural a base de cores parametrizables no deja de ser una manera un tanto primitiva. Además, por el hecho de estar empleando Java la instanciación y parametrización puede resultar demasiado verbosa. La primera objeción es sin duda consecuencia de haber creado un producto orientado a la reconfiguración en tiempo de ejecución. Probablemente, las dificultades de congeniar un estilo de diseño comportamental con el requerimiento de que los circuitos se puedan generar en tiempo de ejecución hubieran hecho este proyecto inabordable. Con respecto a la segunda objeción, viene dada por haber tratado de extender un lenguaje que en un principio sólo estaba pensado para describir software a la descripción de hardware. Es el precio que hay que pagar por usar una plataforma estándar, aunque como se verá más adelante las ventajas que esto supone deberían superar a este inconveniente.

Hay otras dos objeciones menos de fondo que se pueden hacer a JBits. La primera es que sólo se soporte la familia Virtex 'clásica', y no sean soportados dispositivos más modernos. A la hora de escribir este apéndice, Xilinx estaba trabajando en transformar JBits en una herramienta multi-dispositivo, que diera soporte a la familia Virtex-II [Sun03] La última objeción es que, como toda herramienta de investigación, no está ni mucho menos libre de bugs, y su documentación no es todo lo completa que se esperaría que fuera. Es de esperar que estos problemas se vayan solucionando según la herramienta evolucione.

# 2. Flujo de diseño en JBits

El flujo más sencillo de diseño con JBits es el estático, que se describe en la Fig. 129. Se parte de un bitstream, bien vacío o que ya contenga algún diseño, se modifica con JBits, y se vuelve a escribir en forma de archivo .bit. Aunque esta metodología no llega a emplear las características más atractivas de JBits, como son la reconfiguración en tiempo de ejecución o el diseño dinámico, puede sin embargo resultar de utilidad. En general, se beneficiarán de está técnica los diseños que necesiten un control a más bajo nivel del bitstream que el que pueda proporcionar las herramientas convencionales, y aquellos diseños que posteriormente vayan a ser manejados por aplicaciones que hagan

uso del API de JBits. Por ejemplo, en el contexto de esta tesis esta metodología es la que se emplea para añadir los sensores de temperatura a un diseño ya existente, realizado con las herramientas de diseño convencionales, como puede ser un punto caliente en los experimentos de mapa térmico. Pero la operación de los sensores se hace con herramientas más sofisticadas, como la reconfiguración en tiempo de ejecución y el readback.

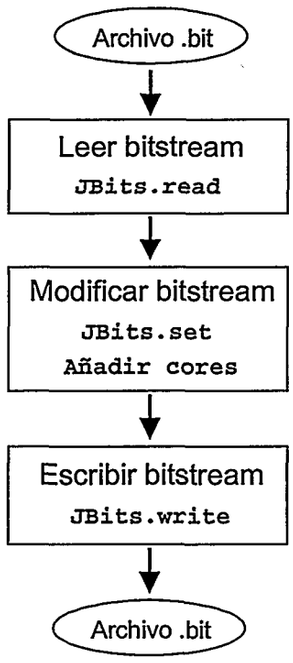



Fig. 129. Flujo elemental de diseño en JBits

Pero la metodología de diseño más adecuada a JBits es la que se muestra en la Fig. 130. Se parte de un bitstream inicial, y sobre esta base se irá modificando el diseño, reconfigurando las partes de la FPGA que hayan cambiado, observando los resultados, y se volverá a modificar el bitstream y se volverá a reconfigurar, y así tantas veces se quiera. Los nuevos circuitos se crearán dinámicamente, en tiempo de ejecución. Esta es una de las ventajas de JBits, pues a diferencia de las herramientas de diseño convencionales, donde el diseño debe ser emplazado y rutado, operación que lleva un tiempo no despreciable, con JBits se crea un diseño ya orientado a su parametrización, y en tiempo de ejecución sólo se definen los parámetros. El diseño es a modo "Lego", mediante bloques que se van ensamblando

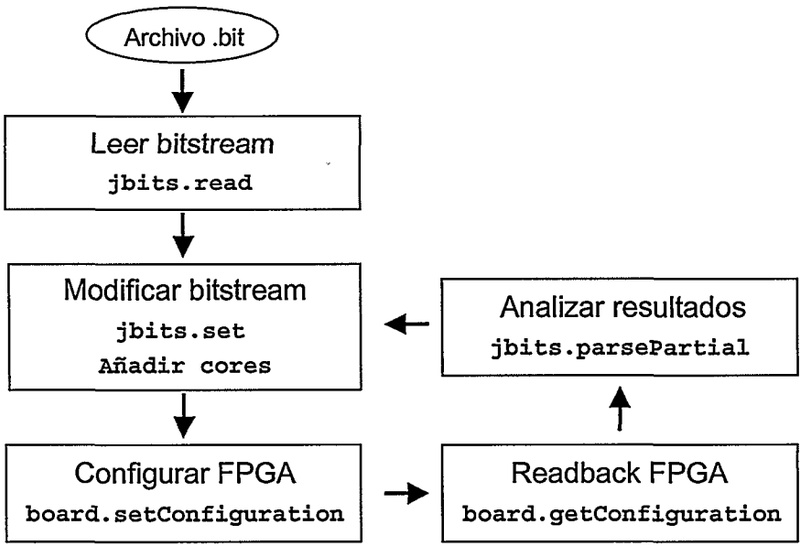



Fig. 130. Flujo de diseño avanzado en JBits

# 3. Diseño de bajo nivel

Como ya se expuso anteriormente, JBits es en primer lugar una herramienta para modificar bitstreams. Para ello ofrece la clase `JBits`, que abstrae el bitstream de una FPGA de la familia Virtex. Esta clase proporciona tres métodos principales, `JBits.set`, `JBits.setIOB` y `JBits.setBram`, que se usan respectivamente para fijar el valor de los bits de configuración en las tres zonas principales del chip: CLBs, IOBs y BRAM. Los prototipos de estos métodos son:

```
void set(int clbRow, int clbColumn, int[][] bits, int val)
void setIOB(int side, int index, int[][] resource, int[] val)
void setBram(int bramRow, int bramColumn, int[][] resource,
             int[] val)
```

Donde `bits` y `resource` indican los bits de configuración que se van a modificar, y `val` son sus nuevos valores. Ambos parámetros se especifican usando constantes definidas en diversas clases; por ejemplo, la clase `com.xilinx.JBits.Virtex.Bits.S0F1` abstrae la configuración del multiplexor en la entrada F1 del *slice* 0 de cualquier CLB. En la siguiente tabla se detallan los recursos del *slice* 0 de un CLB. Existen idénticos recursos para el *slice* 1.

| | |
|---|---|
| Pines de entrada de las LUTs | `S0F1 S0F2 S0F3 S0F4 S0G1 S0G2 S0G3 S0G4` |
| Otros pines de entrada | `S0BX S0BY S0CE S0Clk` |
| Control de las salidas | `S0Control.X S0Control.Y S0Control.YB S0Control.XDin S0Control.YDin` |
| Control del Acarreo | `S0Control.AndMux S0Control.Cin S0Control.XCarrySelect S0Control.YCarrySelect` |
| Otros controles | `YffSetResetSelect YffSetResetSelect Sync SrWeNotInvert LatchMode InvertedSetReset ClockInvert CeInvert ByInvert ByInvert` |

Tabla 20: Clases JBits para la configuración de los recursos del *slice* 0

Un ejemplo de aplicación mínima podría ser el siguiente. Primero se lee un bitstream, se modifica en él el estado de un multiplexor, y por último, se escribe en un nuevo archivo .bit. En particular, lo que se hace es conectar la entrada F1 del *slice* 0 del CLB en la fila 5 y en la columna 4 con la salida X del *slice* 1)

```
JBits jbits = new JBits(Devices.XCV100);
jbits.read("infile.bit");
jbits.set(5, 4, S0F1.S0F1, S0F1.S1_X);
jbits.write("outfile.bit");
```

Si bien esta metodología de diseño puede ser muy útil como una forma de controlar el diseño a muy bajo nivel, mucho más allá de lo que pueden permitir las herramientas de diseño convencionales, o para realizar ingeniería inversa de un diseño, como método de diseño puede resultar extremadamente doloroso. Es por eso que el elemento clave en JBits es el core parametrizable en tiempo de ejecución, que se detalla en el siguiente punto.

## 4. Diseño con RTP cores

Los cores parametrizables en tiempo de ejecución, RTP cores, son cores prediseñados construidos en forma de clases Java. Como están ya prediseñados, en el momento en que se llame a su método `implement` quedará construido el circuito, sin tener que pasar por el emplazado y el rutado con las herramientas convencionales. Los únicos pasos previos que habrá que realizar es instanciar el objeto, definir sus

parámetros y fijar su posición en la FPGA. Esta es la razón para el nombre de parametrizables en tiempo de ejecución, pues son generados dinámicamente sin que sea necesario conocer de antemano sus parámetros, que serán calculados en tiempo de ejecución por el código Java que los instancie.

Todos los cores ocupan un área rectangular de la FPGA, que será dependiente de los parámetros que se hayan definido. De esta manera, el diseño con RTP cores se puede decir que se basa en encajar bloques en la FPGA a la manera de un puzzle o un "Lego". Siguiendo los conceptos de programación orientada a objetos, cada core es una clase de Java, que definen un circuito genérico: p. ej, un multiplicador por constante arbitraria, un contador de n bits... Cada instancia de esa clase será un objeto que se corresponderá con un circuito en particular, dependiendo de los parámetros que se hayan definido en la instanciación. Así, habrá objetos que se corresponderán con un multiplicador por 10, un contador de 16 bits...

## 4.1. Cores disponibles con la distribución de JBits

La distribución de JBits ofrece decenas de cores ya diseñados, que proporcionan desde elementos de la más básica funcionalidad, como LUTs, FFs, cadenas de acarreo... hasta elementos de alto nivel, como filtros FIR, etapas CORDIC, multiplicadores-acumuladores... pasando por los elementos básicos de todo diseño: contadores, sumadores, multiplexores, puertas lógicas...

## 4.2. Utilizando los cores

Para utilizar un core hay que seguir estos cuatro pasos:

1. Instanciar el objeto de la clase de core que se desee y definir sus parámetros.
2. Fijar su posición geográfica
3. Implementarlo (introducirlo en el bitstream de la FPGA)
4. Conectarlo con el resto del diseño.

### *4.2.1. Instanciar y definir los parámetros del core*

Para usar un core lo primero que hay que hacer es instanciarlo como un objeto de la clase que lo abstrae. Los parámetros que definen al core en particular se suelen

especificar en la instanciación. Pero esta no es una regla válida para todos los casos. En realidad existen estas tres posibles alternativas de definir los parámetros:

- Los parámetros se pasan directamente al constructor.
- Se instancia antes un objeto que encapsula los parámetros del core.
- Los parámetros se especifican en llamadas a métodos, antes de llamar a `implement`.

Un ejemplo del primer tipo de posibilidades pueden ser los siguientes prototipo de métodos constructores:

```
CORDIC ( java.lang.String instanceName, Net clk,
         Bus xin, Bus yin, Bus zin,
         Bus xout, Bus yout, Bus zout,
         int mode, int numStages )
Adder ( java.lang.String instanceName, Net clk,
        Bus a, Bus b, Bus out, Net cin )
```

Una segunda posibilidad es crear un objeto cuya utilidad sea sólo servir de almacén para los parámetros, y pasar este objeto al constructor del core. Esto es porque en OOP las estructuras de datos se encapsulan en clases de objetos:

```
CounterProperties CounterProps = new CounterProperties();
CounterProps.setOut_dout(CountOut);
CounterProps.setIn_clk(Clk);
CounterProps.setIn_ce(Net.NoConnect);
CounterProps.setIn_rst(Net.NoConnect);
Counter MyCounter = new Counter("MyCounter", CounterProps);
Offset CounterOffset = MyCounter.getRelativeOffset();
CounterOffset.setVerOffset(Gran.CLB,5);
CounterOffset.setHorOffset(Gran.CLB,5);
MyCounter.implement();
```

La última posibilidad es definir los parámetros mediante llamadas a métodos antes de la llamada a `implement`:

```
AFX50 board = new AFX50("board");
int MyOut = board.addOutput("MyOut", RegOut);
board.setOutputInvertT(MyOut, true);
board.setOutputInvertO(MyOut, false);
board.setConfigureIOBs(true);
board.implement(0, "myout.ucf");
```

Es importante resaltar que sólo por instanciar el core y definir sus parámetros el circuito no está operativo. Es más, todavía no se ha llegado ni a agregarlo al bitstream. Antes hay que asignarle su posición, llamar al método `implement`, y conectarlo con el resto de elementos del sistema.

### *4.2.2. Fijar la posición geográfica del core*

Llegado este punto, se tiene un core instanciado y con sus parámetros ya definidos. Estos parámetros van a fijar el tamaño físico del core (las dimensiones del rectángulo de CLBs que lo va a englobar). Una vez conocido el tamaño del core, se podrá decidir su posición dentro de la FPGA, para lo cual se escogerá un área vacía lo suficientemente grande. Para especificar esta posición geográfica se debe acceder a su clase `Offset` asociada:

```
Offset regOffset =reg.getRelativeOffset();
regOffset.setVerOffset(Gran.CLB,row);
regOffset.setHorOffset(Gran.CLB,col+2);
```

La posición se puede fijar con una precisión (granularidad) de CLB, *slice* o LE. Este es un parámetro que depende de cada core, en unos se podrá fijar con una mayor precisión, y en otros con menos, aunque la inmensa mayoría trabajan con granularidad de CLB.

### *4.2.3. Implementar el core*

Una vez definidos los parámetros del core y su posición en la FPGA, el siguiente paso es naturalmente implementarlo en el dispositivo, o sea, agregarlo al bitstream de configuración de la FPGA. Esto se hace llamando al método `implement`, que a su vez va a llamando a `JBits.set` para ir modificando el bitstream de la FPGA e ir creando en él el core.

### *4.2.4. Conectar el core*

El último paso es conectar las entradas y salidas del core con el resto del sistema. En JBits, los nodos de conexión se abstraen mediante la clase `Net`, y un conjunto de nodos, mediante la clase `Bus`, siguiendo la nomenclatura estándar. Los objetos de la clase `Net` no son nada más que conjuntos de conexiones, que se abstraen con la clase `Port`. Para rutar un nodo se debe llamar a su método `connect`, que a su vez invocará a JRoute, la herramienta de rutado de JBits.

## 4.3. Ejemplo de diseño

### *4.3.1. Aplicación básica JbitsCommandLineApp*

La clase `JBitsCommandLineApp` implementa el código básico para realizar una aplicación que siga el flujo de diseño básico JBits (Fig. 129). Es decir, una aplicación que lea un bitstream, lo modifique, y lo vuelva a escribir. Lo único que hay que hacer es que la aplicación sea una clase derivada de `JBitsCommandLineApp`. Se deberá reescribir el método `main`, e implementar un método `run` donde se encapsularán todas las llamadas a JBits necesarias para crear el circuito. El nuevo método `main` tendrá siempre un aspecto similar al de este código:

```
public static void main(String args[])
{
      tutorial test = new tutorial();
      test.setApplicationName("tutorial");
      test.parseCommandLine(args);
      test.getJBits();
      test.run();
      test.writeBitstream();

}
```

Como se puede ver, justo antes de escribir el bitstream se llama al método `run` para crear el circuito; en el siguiente punto se puede ver un ejemplo de como se debe codificar este método. Los parámetros de línea de comandos que se le pasan a una `JBitsCommandLineApp` son el tipo de dispositivo, el bitstream de entrada y el de

salida. Adicionalmente se puede añadir cuantos se quieran, pero tiene que ser el nuevo código que cree el usuario el que se encargue de gestionarlos.

### *4.3.2. Código para generar un contador y un registro*

El siguiente código genera un contador cuya salida se conecta a un registro. Si se usa una herramienta gráfica de depuración como *BoardScope* se puede ver como con cada pulso de reloj se va incrementando el valor guardado en el registro

```
public void run() {
  try {
    /* especificar JRoute y JBits */
    JRoute jroute = new JRoute(super.getJBits(), System.out);
    Bitstream.setVirtex(super.getJBits(), jroute);
    /* especificar el CoreOutput */
    CoreOutput.generateBitstream(true);
    CoreOutput.generateSYM(false);
    CoreOutput.generateXDL(false);
```

Estas primeras líneas son necesarias para especificar sobre que objetos `JRoute` y `JBits` se va a trabajar, y para indicar los tipos de salidas que se habilitan en la generación de cores. Como objeto `JBits` se utilizará el que corresponde con el bitstream pasado como segundo parámetro, y se obtiene haciendo una llamada al método `getJBits` de `JBitsCommandLineApp` (la clase de la que se deriva la aplicación). Por otro lado, sólo se habilita la salida que genera el bitstream, y se deshabilitan las de depuración. A continuación se definen los nodos que va a tener el circuito: el de reloj, y la salida del contador:

```
    /* nets y buses */
    Net Clk = new Net("Clk",null);
    Bus CountOut = new Bus("CountOut",null,CounterSize);
```

Instanciar e implementar el buffer de reloj es muy sencillo, pues su posición física se especifica en la llamada a `implement`.

```
    /* core de reloj. Utiliza GCLK1 */
    Clock MyClock = new Clock("mainClock", Clk);
```

```
MyClock.implement(1);
```

Sin embargo, el contador necesita mucho más código, pues primero hay que crear un objeto de la clase `CounterProps` para especificar sus conexiones. Y por otro lado, antes de llamar a `implement` hay que fijar su posición dentro de la FPGA

```
/* propiedades del contador */
CounterProperties CounterProps = new CounterProperties();
CounterProps.setOut_dout(CountOut);
CounterProps.setIn_clk(Clk);
CounterProps.setIn_ce(Net.NoConnect);
CounterProps.setIn_rst(Net.NoConnect);
/* contador */
Counter MainCounter = new Counter("Contador", CounterProps);
Offset MainCounterOffset = MainCounter.getRelativeOffset();
MainCounterOffset.setVerOffset(Gran.CLB,1);
MainCounterOffset.setHorOffset(Gran.CLB,1);
MainCounter.implement();
```

Los registros siguen el esquema más estándar: instanciar el objeto, asignarle una posición, y por último, llamar a implement. En este caso se va a usar su salida, por lo que se ha optado por el constructor de tres parámetros: nombre, nodo de reloj y bus de entrada.

```
/* registros */
Register DummyReg =  new Register("DummyReg", Clk, CountOut);
Offset DummyRegOffset = DummyReg.getRelativeOffset();
        DummyRegOffset.setVerOffset(Gran.CLB,1);
        DummyRegOffset.setHorOffset(Gran.CLB,3);
DummyReg.implement();
```

El último paso es siempre rutar las conexiones entre módulos, lo que se hace con llamadas a `connect`.

```
/* por ultimo, conectarlo todo */
Bitstream.connect(Clk);
Bitstream.connect(CountOut);
```

```
    Bitstream.connect(SR);
  }
  catch (Exception e) {
    e.printStackTrace();
    System.exit(-1);
  }
} /* run() */
```

Como comentario final, es importante notar que todo el código está dentro de un `try`, porque la mayoría de llamadas a JBits pueden generar excepciones.

## 4.4. Como construir un core

Los cores en JBits pueden ser derivados, que están formados por la unión de otros cores, o primitivos, cuando no se basan en ningún core, y son creados desde el principio usando los recursos básicos de la FPGA. Dependiendo del tipo de core que se vaya a crear, se seguirá una metodología distinta. Pero en ambos casos hay una serie de pasos comunes, que se detallan a continuación.

### *4.4.1. Acciones comunes en la creación de un core derivado o primitivo*

Todo core debe implementar al menos estos cuatro métodos, que se usan para calcular de antemano cual va a ser su tamaño, sin necesidad de instanciarlo:

```
public static int calcHeight(PARAMETERS)
public static int calcWidth(PARAMETERS)
public static int calcHeightGran(PARAMETERS)
public static int calcWidthGran(PARAMETERS)
```

Los parámetros que se le pasarán a estos métodos se corresponden con los parámetros que tendrá el core que se desea instanciar (p.ej. en el caso de un contador, número de bits).

Como toda clase de Java, debe implementar uno (o varios) constructores. Típicamente, al constructor se le pasarán los parámetros del core que se va a instanciar. Pero como ya se vio antes, existen otras alternativas para definir estos parámetros. Lo que siempre se le debe pasar al constructor es el nombre de la instancia, en forma de `String`.

El constructor básicamente debe realizar las siguientes acciones:

- Fijar el nombre de la instancia, llamando al constructor de la clase padre
- Definir las conexiones externas del core, como instancias de `Port`
- Fijar el tamaño del core, mediante llamadas a `setsize`
- Por último, todos los cores deben tener el método `implement`. Este es el método que construye realmente el core, creando su circuito en el bitstream de la FPGA. Es aquí donde existirán diferencias dependiendo de que el core sea primitivo o derivado.

### *4.4.2. Construir un core derivado*

La metodología es completamente similar a la expuesta antes para crear un circuito basado en cores; la única diferencia es que ahora los cores se deben añadir usando el método `AddChild` de `RTPCore`.

Por otro lado se debe resolver la interfaz del core con el exterior. En JBits, la clase `Port` es la que se usa para abstraer las conexiones externas de un core. Para crear los *ports* se usarán las señales externas (`Net` o `Bus`) que se conectan con el core; normalmente estas señales serán pasadas como parámetros en el constructor de la clase que define el core. Los *ports* de entrada se generarán con `newInputPort`, y los de salida, con `newOutputPort`. En el siguiente ejemplo se muestra el código que creará los puertos para un circuito muy sencillo, un flip-flop tipo D:

```
private Port clkPort, dPort, qPort;
public flip_flop_d(Net clk, Net d, Net q)
{
   ...
   clkPort = newInputPort("CLK", clk);
   dPort = newOutputPort("D", d);
   qPort = newOutputPort("Q", q);
}
```

De esta manera se especifican cuales son las conexiones de los puertos en el lado externo del core; para definirlas en su lado interno se debe utilizar el método `setIntSig` de `Port`. Siguiendo el ejemplo anterior,

```
public void implement() throws CoreException
{
      Net clkNet = newNet("clk");
      clkPort.setIntSig(clkNet);
      Net dNet = newNet("d"); Net qNet = newNet("q");
      dPort.setIntSig(dNet); qPort.setIntSig(qNet);
      ...
}
```

Hay que destacar que los nodos se han creado usando el método `newNet` de RTPCore; así se indica que son internos al core.

### *4.4.3. Construir un core primitivo*

Los cores primitivos se construyen mediante llamadas de bajo nivel, usando `jbits.set`, `jbits.setBRAM` y/o `jbits.setIOB`. De esta manera se van configurando todos los elementos de la FPGA a partir de los cuales se forma el core. El rutado también se puede hacer a bajo nivel, usando `jbits.set`, pero es en general una tarea demasiado compleja, a lo que hay que unir también la falta de documentación. Es por esto que lo más recomendable es usar la herramienta propia de rutado, JRoute. Para ello, hay que declarar primero los *pines* que se desean conectar, construyendo nuevos objetos de la clase `Pin`. En la llamada al constructor de esta clase se especifica que *pin* en particular queremos usar mediante constantes definidas en la clase `CenterWires`. Por ejemplo,

```
Source = new Pin(Pin.CLB, 4, 5, CenterWires.Slice_X[1]);
Sink = new Pin(Pin.CLB, 2, 3, CenterWires.SliceF1[0]);
```

El primer *pin* (`Source`) se corresponde con la salida X del *slice* 1 del CLB situado en la columna 5 y la fila 4; y el segundo *pin* (`Sink`) es la entrada F1 del *slice* 0 del CLB en la columna 3, fila 2.

Una vez definidos los *pines*, rutar un camino entre ellos es muy sencillo: sólo hay que hacer una llamada a `Bitstream.connect`, con parámetros el *pin* origen y el *pin* destino. Siguiendo el ejemplo anterior:

```
Bitstream.connect(Source, Sink);
```

Respecto a los puertos de E/S, las conexiones externas se especifican de igual manera que en los cores derivados, usando `newInputPort` o `newOutputPort` sobre los nodos que se han pasado como parámetros del constructor. Sin embargo, en los cores primitivos no se suelen definir nodos internos, por lo que para definir los puntos de conexión de los puertos lo más apropiado es asociarles un `Pin` o una matriz de ellos, usando el método `setPin` de `Port`. Por ejemplo,

```
public void implement() throws CoreException
{
    Pin[] clkPins = new Pin[2];
    ...
    clkPins [0] = new Pin(Pin.CLB,1,1,CenterWires.SliceClk[0]);
    clkPins [1] = new Pin(Pin.CLB,2,1,CenterWires.SliceClk[0]);
    clkPort.setPin(clkPins);
}
```

# 5. Depuración de circuitos

Como se ha venido indicando hasta ahora, JBits es una metodología de diseño que se separa de los cauces estándares, por lo tanto en un principio deberá también proveer sus propias herramientas de depuración. Aunque, como se verá al final de este punto, también va a ser posible usar simuladores convencionales.

## 5.1. BoardScope / WaveformViewer

La herramienta principal para la depuración con JBits es *BoardScope*. Es una aplicación escrita en Java, y que a través del API de JBits permite realizar las siguientes tareas:

- Permite cargar bitstreams o cores
- Muestra el estado actual de la FPGA

- Visualiza los cores que se han cargado
- Permite mandar ciclos de reloj, hacer readbacks, mandar reset...

El aspecto de la interfaz de usuario puede verse en la Fig. 131. Proporciona también una interfaz sobre dos herramientas: DDT y *WaveformViewer*. La primera es intérprete de comandos para el manejo interactivo de cores; no es de especial utilidad. Sin embargo, la segunda es la que es de más interés. *WaveformViewer* es un visor interactivo de formas de onda, que va a permitir al usuario mandar pulsos de reloj al circuito y observar visualmente sus resultados. Una muestra de esto puede observarse en la Fig. 132.

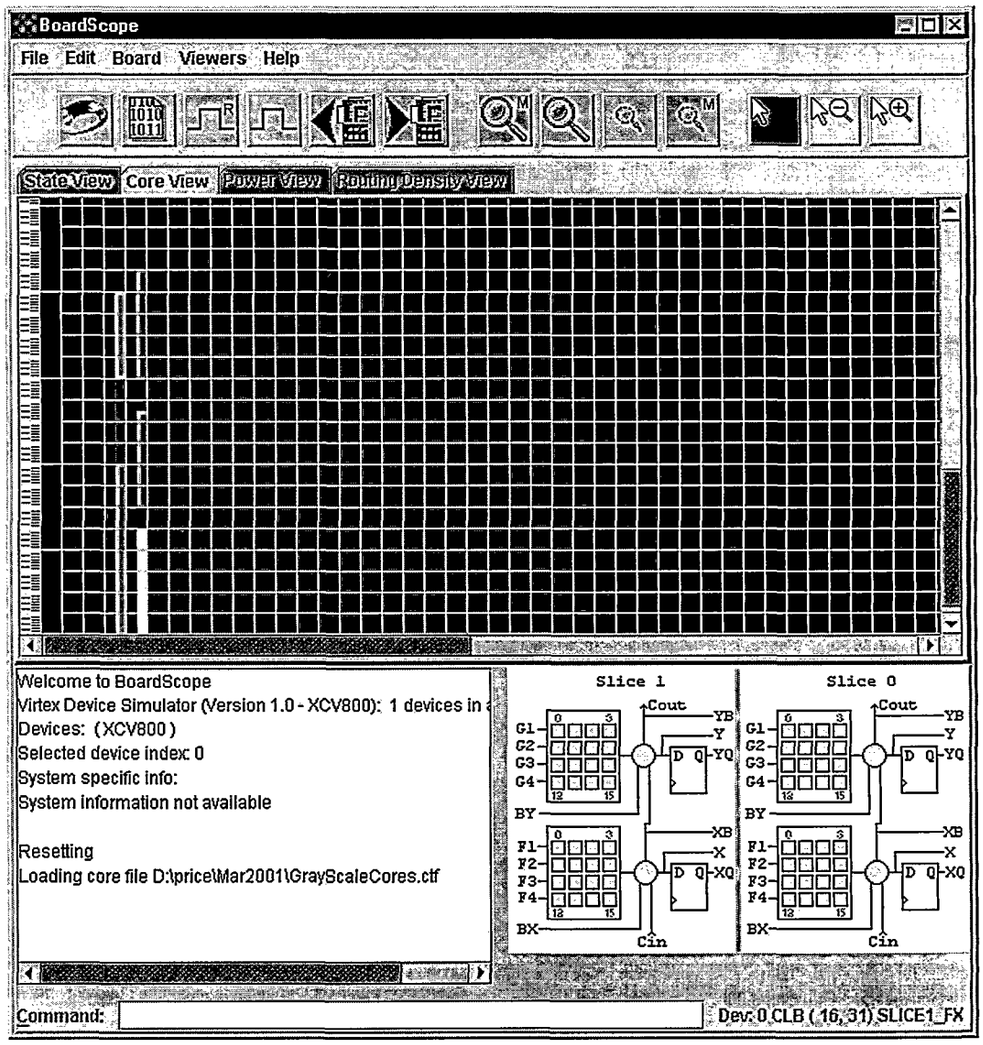



Fig. 131: Herramienta *BoardScope*

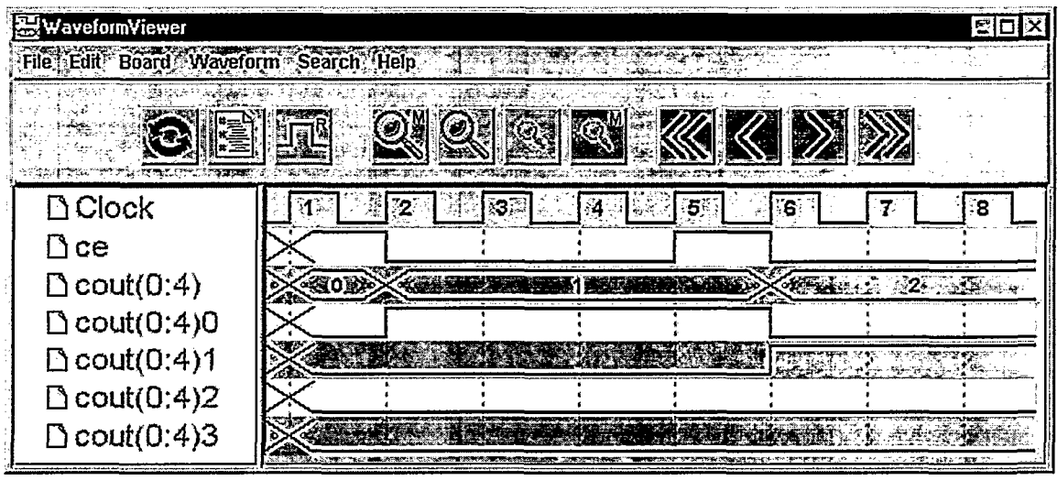



Fig. 132: Herramienta WaveformViewer

*BoardScope* es la herramienta más visual, pero no significa que sea el instrumento más potente para la depuración. La manera más efectiva de depurar el circuito es escribir código JBits que haga uso de las partes del API diseñadas para depuración. Esto no es algo insólito, igualmente la mejor manera de depurar un circuito descrito en VHDL es crear un banco de pruebas también escrito en VHDL.

## 5.2. VirtexDS

El primero de estos elementos es *VirtexDS*. Es un simulador de una tarjeta con una FPGA, de tal manera que los diseños se pueden descargar y probar sobre este simulador sin correr riesgos de que un error pueda dañar el hardware. Las operaciones que permite este simulador son las siguientes

El comportamiento del circuito puede observarse usando readback, como en cualquier FPGA real. Adicionalmente, se puede fijar el valor de cualquier punto de la FPGA o leer su valor, sin la limitación de que sólo se puede acceder a los registros que impone el readback

## 5.3. Cores para testeo

JBits proporciona dos cores que permiten realizar los pasos básicos de la comprobación: la generación de vectores de test y la captura de resultados

### 5.3.1. TestInputVector

Este core permite generar vectores de test, de un ancho de hasta 64 bits, y con una profundidad variable, pero siempre múltiplo de 16. Esto es porque para implementarlo se usan los CLBs configurados como registros de desplazamiento (SRL16); así, el core ocupa 1 *slice* por cada dos bits de ancho y 16 de profundidad.

El constructor del core tiene este prototipo:

```
TestInputVector(java.lang.String instanceName,
                int depth, Net clk, Bus dout)
```

El core es síncrono, con cada pulso de reloj saca un nuevo vector por el `Bus dout`. Los valores de los vectores se especifican en la llamada a `implement`, bien como un vector o referenciando el archivo que los contiene.

### 5.3.2. CoreTester

Este core utiliza BlockRAM tanto para los generar vectores de test como guardar los resultados devueltos por el circuito bajo prueba. Tanto el ancho como la profundidad son parametrizables, con la única limitación de que sólo usa un bloque de memoria para las entradas y otro para las salidas, por lo que el ancho de los datos multiplicado por el número de muestras no debe superar 4096 (que es el tamaño de los bloques de memoria en Virtex).

La señal de reloj que se debe usar y los buses de entrada y salida se especifican en un objeto de la clase auxiliar `CoreTesterParameters`. Otros parámetros que se pueden definir son la posición de las BlockRAMs que se van a usar, y el número de ciclos de reloj por muestra. Este objeto se pasa como parámetro al constructor de `CoreTester`. Por último, los valores de los vectores de test se definen en la llamada a `implement`.

### 5.3.3. XDL

A parte de generar el archivo .bit, JBits también puede generar archivos de tipo XDL (*Xilinx Design Language*). Este es un formato intermedio que puede traducirse a archivos NCD; este formato es el que emplea el editor de FPGAs, y también es el punto de partida para obtener modelos post-layout VHDL o Verilog de la FPGA. Esta capacidad de JBits

es muy relevante, pues da acceso a las herramientas estándar de depuración. Sin embargo, la generación de archivos XDL es uno de las apartados menos depurados de JBits, al menos en el momento de escribir este apéndice. Por ejemplo, no soporta el uso de IOBs, se pierde el rutado de la línea de reloj y de las BRAMs, se pueden producir errores en la generación del archivo XDL,...

Sólo se pueden generar archivos XDL desde las aplicaciones que sigan el API *CoreTemplate* (o sea, que usen los cores que vienen con la distribución, o cores nuevos si se ajustan a esta especificación). En un principio esto no representa ninguna limitación, pues la manera estándar de diseñar con JBits es usar cores. Justo antes de instanciarlos se debe incluir una llamada como esta:

```
CoreOutput.generateXDL(true);
```

Que indica que se desea generar un archivo XDL. Cuando el diseño ya esté completo (cores instanciados e implementados, nodos rutados), con la siguiente llamada se escribe el archivo:

```
CoreOutput.writeXDLFile("archivo.xdl");
```

Una vez que se tiene el archivo XDL, utilizando la utilidad `xdl.exe` (disponible en las herramientas estándar de Xilinx) se consigue el archivo NCD.

## 6. Acceso al hardware

Una de las grandes ventajas de JBits es que todas las tareas necesarias para realizar una configuración total o parcial de la FPGA están perfectamente encapsuladas, resultando muy sencillas de codificar.

El primer paso para configurar una FPGA es instanciar un objeto de la clase que abstrae el tipo de tarjeta en la que está embebida. Una vez hecho esto, con una llamada al método `connect` queda establecida la comunicación. Cada tarjeta puede tener una o varias FPGAs; esto es apropiado para sistemas multi-dispositivo, tales como ordenadores a medida (*custom-computers*). Una vez que se ha establecido la conexión con el hardware, las acciones más normales serán inicializar la tarjeta (método `reset`), escribir datos de configuración (método `SetConfiguration`) y leerlos (método

`GetConfiguration`). Por último, la desconexión del hardware se hará mediante el método `disconnect`.

Para obtener el comando de configuración que se enviará a la FPGA habrá que llamar al método `getPartial` sobre el objeto JBits en el que estemos trabajando. Esta llamada devolverá un comando que configurará todos los *frames* de la FPGA que estén 'sucios', es decir, que hayan cambiado desde la última llamada. Así la reconfiguración parcial se maneja de una manera completamente transparente, sólo se reconfiguran las partes de la FPGA que sean nuevas. Si se quiere forzar una reconfiguración parcial de la FPGA, llamando al método `clearPartial` se señalan todos los *frames* de la FPGA como 'sucios'.

Por último hay que mencionar que las tarjetas normalmente no sólo contienen FPGAs, es habitual que dispongan de memoria externa y de un generador de reloj programable. Para ello JBits proporciona métodos que permiten acceder a la memoria (`setRAM`, `getRAM`) y programar el generador de reloj (`setClockFrequency`, `clockOn`, `clockOff`, `clockStep`).

## 6.1. Ejemplo de código para configurar la FPGA

A continuación se muestra un ejemplo que carga un bitstream en una FPGA, a continuación modifica una LUT y vuelve a reconfigurar la FPGA, cambiando sólo el frame que ha cambiado:

```
custom myBoard = new custom();
myBoard.connect(null,0);
JBits jbits= new JBits(Devices.XCV50);
jbits.readPartial("test.bit");
myBoard.setConfiguration(0,jbits.getPartial());
jbits.set(row,col,LUT.SLICE0_G,
Util.InvertIntArray(Expr.G_LUT("1")));
myBoard.setConfiguration(0,jbits.getPartial());
```

Como ya se comentó en el punto anterior, el primer paso siempre es instanciar un objeto de la clase de la tarjeta que estamos usando. En este caso instanciamos `myBoard`, que representará una tarjeta del tipo `custom` (una tarjeta hecha a medida por

el usuario). Y a continuación se establece la conexión con el hardware, en la llamada a `connect`. En este caso es una conexión local, por eso el nombre del servidor queda a `null`.

Seguidamente se crea un objeto `JBits` correspondiente con el tipo de dispositivo que se va a emplear, en este caso una XCV50. Y se lee el bitstream ya existente desde `test.bit`, usando `JBits.readPartial`. Este método también soporta reconfiguración parcial, de tal manera que sólo cargará desde el archivo los *frames* que difieran. En este caso, como se acaba de crear el objeto JBits, cualquier carga actualizará todos los *frames*, señalándolos como 'sucios'. Para asegurarse, y aunque no es necesario, se vuelve a fijar todos los contenidos de la FPGA como nuevos en una llamada a `JBits.clearPartial`.

Para realizar la primera configuración se hace una llamada al método `setConfiguration` del objeto que representa la tarjeta que se está usando, `myBoard`, utilizando como datos de configuración los que devuelve `JBits.getPartial`.

Por último, utilizando `JBits.set` se realiza una modificación a bajo nivel de una tabla de lookup, y se vuelve a realizar una configuración de la FPGA. En este caso sólo se reconfigurará los 16 *frames* correspondientes con la LUT modificada, pues el resto de contenidos no habrán cambiado.

## 6.2. XHWIF

XHWIF (*Xilinx Hardware Interface*) es la herramienta que implementa las comunicaciones con el hardware descritas en el punto anterior. Permite manejar las tarjetas bien localmente, desde el propio ordenador en el que esté conectadas, o remotamente, usando TCP/IP. Para gestionar el manejo remoto incluye un programa servidor que será el que se conecte con el hardware, mientras que en el extremo remoto el código que se ejecutará será exactamente igual al que se ha mostrado en el ejemplo anterior, con la salvedad de que en este caso el objeto que representará la tarjeta será de la clase `xhwifnet`, indicando que es un acceso remoto. En la llamada al método `connect`, se deberá además especificar el nombre de la máquina a la que está conectada la tarjeta, y el puerto en el que se ha enganchado el servidor XHWIF.

Para ejecutar el servidor sólo hay que ejecutar un comando del tipo:

```
java XHWIFServer -tarjeta
```

## 6.3. Readback

De la misma manera que JBits implementa muy eficientemente los mecanismos de configuración de la FPGA, el readback está también bien resuelto. El funcionamiento básico es similar a la configuración, como se puede ver en el siguiente ejemplo:

```
/* Mandar un comando de readback */
readbackCmd = ReadbackCommand.getClbConfig(deviceType);
board.setConfiguration(device, readbackCmd);
/* Sacar los datos de readback de la FPGA*/
readbackData = board.getConfiguration(device,
                    ReadbackCommand.getReadLength()*4);
/* Leer los datos y meterlos en un objeto jbits */
jbits.parsePartial(readbackCmd, readbackData);
```

La única diferencia es que el readback se realiza en dos fases. Primero se manda el comando de readback usando `setConfiguration`, luego se leen los datos en sí del readback mediante una llamada a `getConfiguration`. Una vez leído el contenido de la FPGA, el bitstream almacenado en el objeto JBits se actualiza mediante una llamada a `parsePartial` sobre los datos devueltos por `getConfiguration`.

La principal utilidad del readback será conocer el estado de las señales internas, ya sea para depurar el circuito o para conocer el resultado de una operación. Para ello, la clase `State` permite conocer el valor de una variable entera que ha sido almacenada en un conjunto de registros de la FPGA. Sólo hay que definir donde está almacenado cada bit de esta variable, y automáticamente con el método `int` devolverá su valor entero.

# 7. Integración con las herramientas convencionales

Como ya se ha comentado en la introducción, es posible mezclar en un mismo diseño circuitos creados con JBits con otros generados con las herramientas convencionales.

JBits proporciona dos mecanismos para facilitar esta interacción: el core `BlackBox` y los *anticores*.

## 7.1. Pasos previos

El primer detalle que hay que tener en cuenta al hacer diseños mixtos con JBits y las herramientas convencionales de Xilinx son los distintos sistemas de coordenadas que usan uno y otro:

```
ColJBits = ColFPGA - 1
FilaJBits = NumeroDeFilas - FilaFPGA
```

Además, si se usa en JBits un bitstream que ya contiene nodos rutados, es necesario que JBits descubra esos caminos, de tal manera que no los sobrescriba al rutar nodos bajo JBits. Para ello, se debe utilizar siempre un código similar a este:

```
import com.xilinx.JRoute2.Virtex.ResourceFactory;

...

ResourceFactory rf = ResourceFactory.getResourceFactory(jbits);

rf.fillResourceFactory();
```

Donde `jbits` contiene el bitstream no vacío que se ha leído. El proceso de descubrir las rutas en la FPGA puede ser bastante largo, en especial para FPGAs grandes y/o muy llenas. Por otro lado, por los experimentos realizados en esta tesis no parece que sea capaz de descubrir todos los caminos, al menos en la versión actual de JBits (2.8) y anteriores.

## 7.2. Usando cores Blackbox

Estos cores se emplean para indicar que hay partes en el bitstream original que no están vacías, que están ocupadas por lógica creada usada con las herramientas convencionales. Básicamente, su única funcionalidad es añadir un *tag* a estos CLBs, para que JBits los pueda identificar como ocupados y no se sobrescriban.

Si se desea una funcionalidad más compleja, que incluya la posibilidad de conectar el diseño JBits con el convencional, se puede crear un nuevo core como derivado de esta clase. Será necesario codificar un nuevo constructor, al que se le pasen como

parámetros los nodos de interfaz. Este constructor lo único que hará será crear los puertos para estos nodos, y asignarles sus conexiones tanto externas como internas (como ya se vio en el apartado de cómo crear cores primitivos). Para saber a que pines se deben dirigir las conexiones internas de los puertos se deberá recurrir a los informes proporcionados por las herramientas convencionales. Como es lógico, no será necesario codificar de nuevo el método `implement`, pues la implementación del core ya se habrá hecho con las herramientas convencionales.

## 7.3. Usando *anticores*

Los *anticores* permiten un flujo de diseño más complejo que el pueden ofrecer los cores `Blackbox`. Lo que proporcionan es la posibilidad de empezar diseñando con JBits, continuar con las herramientas convencionales, y luego volver otra vez a trabajar con JBits.

El proceso comienza con un diseño que contiene un core; lo que se desea es usar este core como parte de un circuito generado con herramientas convencionales. Para ello, se genera un *anticore* de este core inicial, mediante esta llamada:

```
CoreOutput.makeAntiCore(core_inicial);
```

Que creará una netlist EDIF con el *anticore*, una plantilla de instanciación para VHDL y un código java que se utilizará más adelante para eliminar el *anticore*.

Esta netlist EDIF se usará para instanciar el *anticore* dentro de un diseño convencional (para lo que se podrá usar la plantilla VHDL). El *anticore* contiene una lógica que rellena los CLBs usados por el core original, y además los rodea con una maraña de rutas, de tal manera que el área que ocupaba el core JBits no pueda ser utilizada por el diseño hecho con las herramientas convencionales. La utilidad de este relleno es que, cuando se elimine, quedará un 'hueco' libre donde volver a poner el core original. Si no existiese ese relleno, la lógica generada por las herramientas convencionales ocuparía el área del core JBits, que ya no se podría volver a insertar. Y no vale con poner una directiva PROHIBIT, porque no sólo hay que asegurar que no se ocupan los CLBs, sino también de que se deja espacio para el rutado.

Una vez generado el bitstream de manera estándar, se lee en JBits y se vuelve a poner el core en su posición original. Para ello, primero hay que eliminar el *anticore*; para

ello se puede usar el código java que se creó al generarlo. Este programa reprograma los CLBs, para eliminar la lógica, y deshace todo el rutado. Una vez eliminado el *anticore*, se puede volver a instanciar el core en su posición original, y así se tendrá ya el circuito completo: el core JBits inmerso en el circuito creado con las herramientas convencionales, y además interactuando con él. Ya por último, puesto que el bitstream esta siendo editado en JBits, se podrían hacer modificaciones adicionales, como añadir nuevos cores o variar partes de la configuración, como contenidos de las tablas de lookup.

Toda este proceso se resume en la siguiente figura:

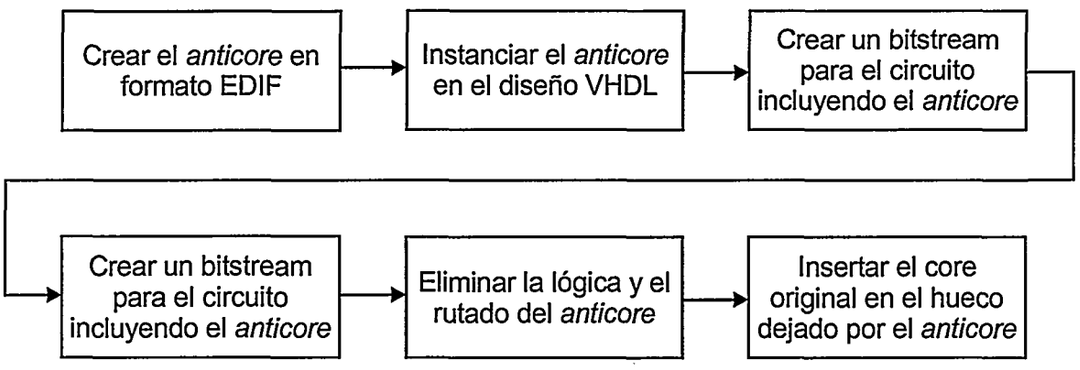



Fig. 133: Pasos para utilizar un RTP core en un diseño convencional usando anticores

# Apéndice B.

# Detalles técnicos adicionales

En este apéndice se ofrecen detalles técnicos adicionales tanto de los sensores en sí como de los montajes usados para calibrarlos. Aunque esta información no será de gran utilidad para el lector en general, si que puede ser interesante para quien esté interesado en seguir investigando en este tema, y tenga que repetir los experimentos presentados en esta tesis, o hacer otros similares.

En la sección 1 se describe la instrumentación usada en la calibración de los sensores para las familias XC3000 y XC4000, que fueron presentados en el capítulo 4. En la sección 2 se presenta la tarjeta empleada para calibrar los sensores basados en Virtex (capítulo 5), y se detalla la interfaz que se ha diseñado para acceder a la FPGA a través de JBits. Por último, en la sección 3 se incluye el código JBits del sensor dinámicamente insertable en la FPGA descrito en el capítulo 5.

## 1. Instrumentación empleada en los experimentos con XC3000 y XC4000.

Para hacer las medidas que se presentaron en el capítulo 4, con FPGAs de la familia XC3000 y XC4000, se utilizó una instrumentación construida a medida. Este equipo estaba basado en un microcontrolador de la familia 68HC11 de Motorola, al que se le conectó una FPGA del tipo XC3130, donde se implementaron los contadores de frecuencia. Su aspecto puede verse en la Fig. 134, donde se ve que estaba compuesto por dos placas, la primera (azul) no era más que un sistema estándar basado en HC11, mientras que la segunda (verde) era la que implementaba todos los circuitos digitales y

analógicos que hacían falta para hacer las experimentos. Todos los circuitos digitales necesarios para medir los osciladores estaban implementados en la FPGA antes mencionada; se puede ver claramente en la fotografía, en un zócalo PLCC84. Por otro lado, esta segunda placa incluía los circuitos analógicos necesarios para hacer las medidas de los diodos de enclavamiento, o sea, polarización y acondicionamiento de señal. La conversión A/D se realizaba en el propio HC11. Ambas placas fueron grapinadas.

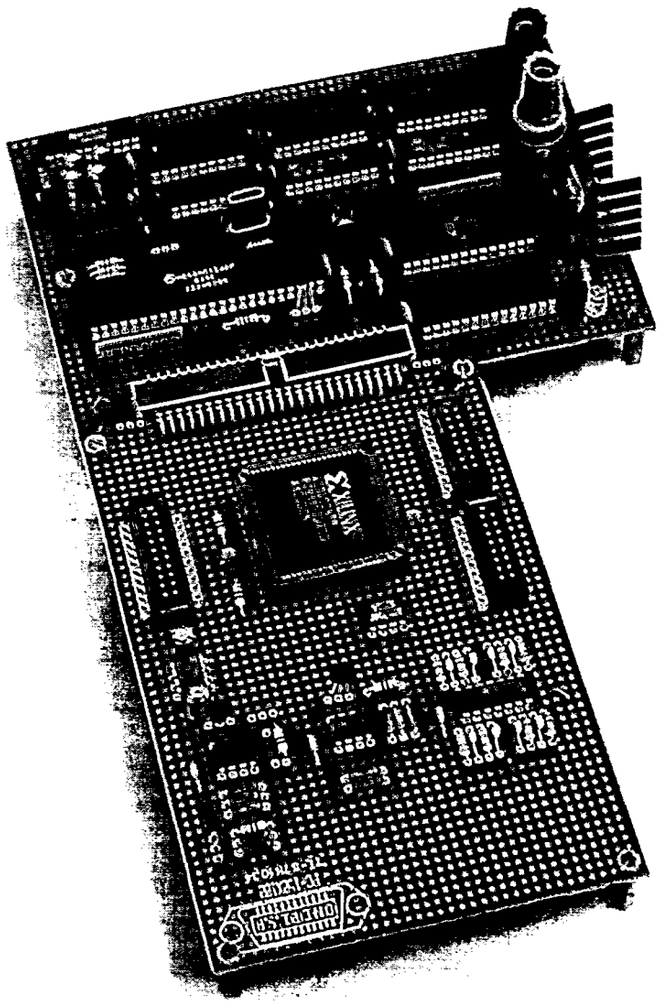


Fig. 134: Fotografía de las placas empleadas para los experimentos con XC3000 y XC4000

En la Fig. 135 puede verse un diagrama de la placa del microcontrolador; no era más que un sistema estándar, con 8 KB de RAM, 8 KB de EEPROM, dos puertos de E/S de 8 bits, un puerto para LCD, una interfaz RS-232 y un puerto de expansión donde se conectaba la otra placa. Para realizar la decodificación de direcciones y otra lógica adicional se empleó una PAL del tipo PALCE16V8, y para desmultiplexar el bus de datos/direcciones se utilizó un latch tipo 74HCT573. La frecuencia de operación era de 1 MHz por limitaciones en la temporización del acceso al LCD. Por último, y puesto que se usan los conversores A/D del microcontrolador, se incluyó una referencia de tensión de 4,096 V del tipo LM9140.

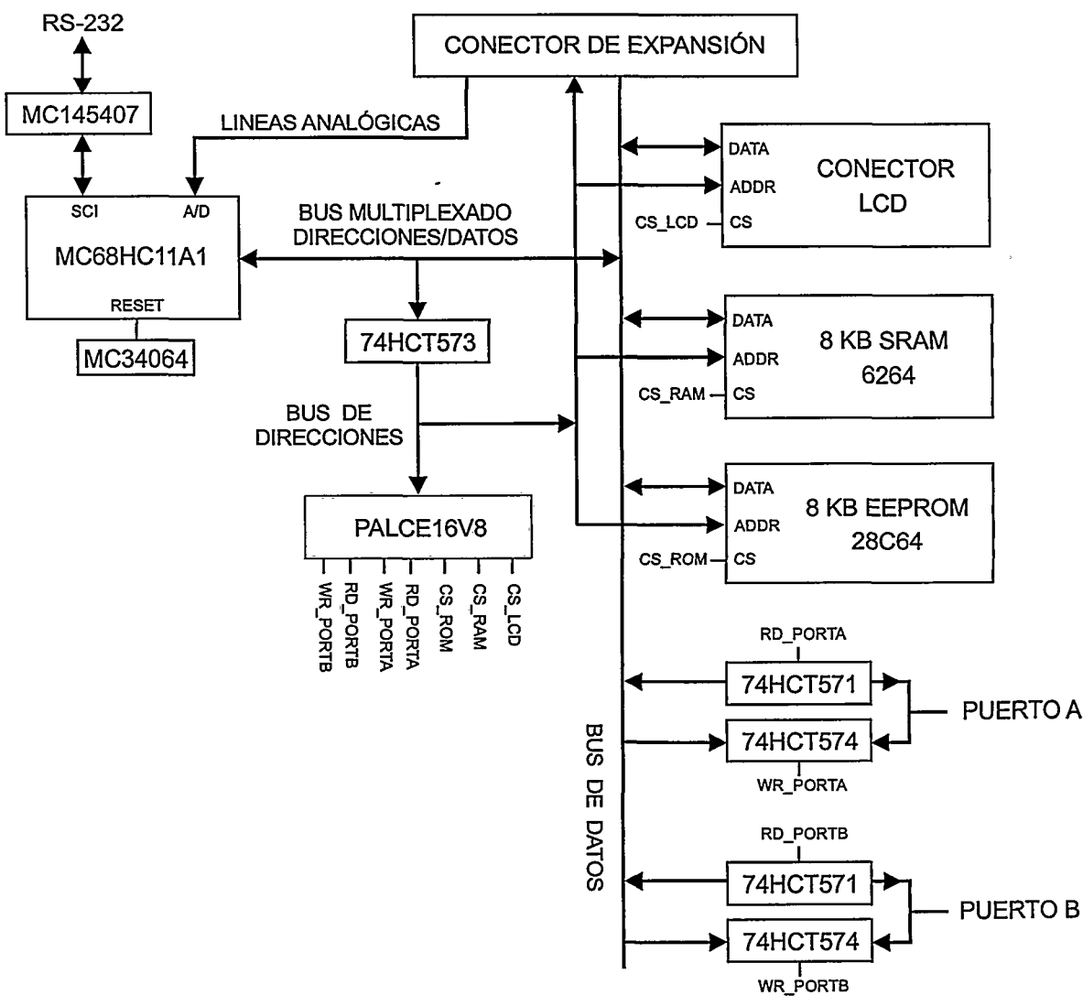



Fig. 135: Esquema de la placa con microcontrolador HC11

Por otro lado, en la FPGA de la placa adicional se implementó el contador de frecuencia para medir los osciladores, y un multiplexor que permitía controlar hasta 16 de ellos, aunque como ya se vio nunca se emplearon más de 4 simultáneamente. La FPGA se programaba a través de una EEPROM serie. La parte analógica consistía del por un lado del circuito de polarización del diodo, que como se pudo ver en la Fig. 41 necesitaba de una tensión de –5 V. Esta tensión se obtenía a partir de un inversor tipo 7660, y luego se estabilizaba con un regulador del tipo LT1009. Una vez recogida la señal del diodo, se le hacía un acondicionamiento básico: compensación de offset y amplificación, y luego se pasaba por un filtro paso bajo de segundo orden para reducir el ruido que se hubiese podido acoplar en los cables.

Las FPGAs sobre las que se hacían las medidas se montaban sobre zócalos PLCC84 en placas de reducido tamaño, también grapinada. Estas placas eran muy sencillas, sólo contenían la FPGA, unos pocos pull-ups y condensadores de desacoplo, un conector para descargar el bitstream, un búfer 74HC125 para aislar la capacidad parásita de los cables, y los conectores para manejar los sensores. En la siguiente figura se puede ver el aspecto de la placa que se utilizó con la XC3030.

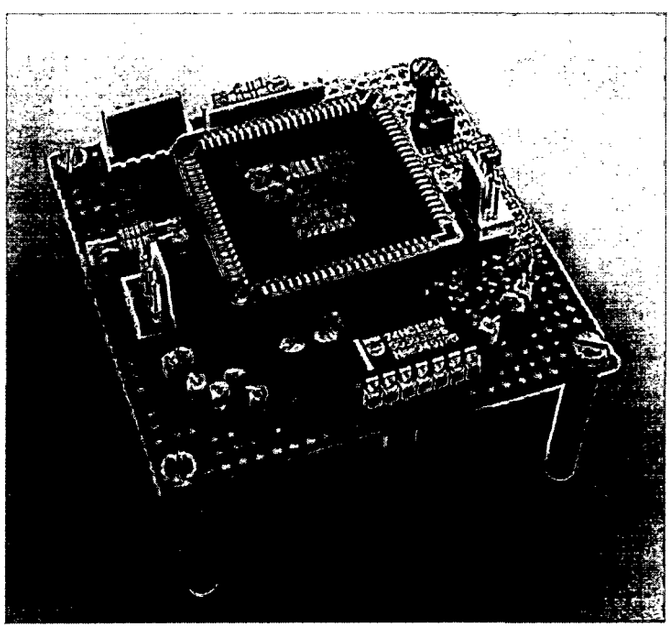


Fig. 136: Fotografía de la placa empleada para calibrar los sensores implementados en la XC3030

# 2. Interfaz con la placa AFX

Para realizar todos los experimentos con FPGAs de la familia Virtex se empleó una tarjeta de prototipado de Xilinx, la AFX PQ240-100 [Xil99d]. Ya se mencionó que esta tarjeta puede parecer muy pobre, porque no tiene ningún componente adicional, pero esa era precisamente la característica que se estaba buscando, porque cualquier otro dispositivo que pudiera haber en la placa interferiría en las medidas de consumo. Su diagrama de bloques, obtenido del manual de usuario, se puede ver en la siguiente figura:

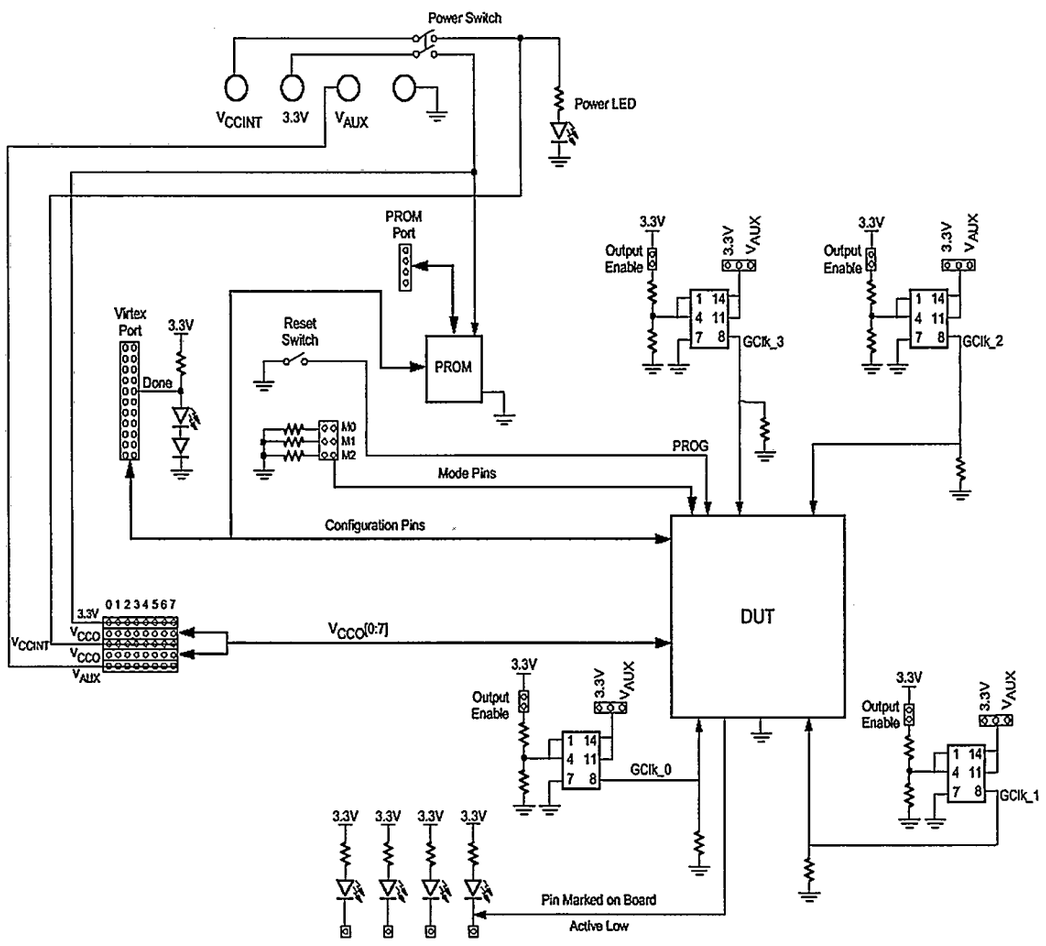



Fig. 137: Esquema de la placa Xilinx AFX (obtenido de [Xil99d])

Básicamente, la tarjeta tiene unas bananas de alimentación, un conector para programar la FPGA, un zócalo para una EPROM de configuración, zócalos para cuatro osciladores (uno por cada línea global de reloj en Virtex), cuatro LEDs, jumpers de configuración (principalmente selección de tensiones en los bancos de E/S y modo de configuración: M0, M1 y M2), y por último, un área de grapinado. En la siguiente figura se puede ver una fotografía de la tarjeta.

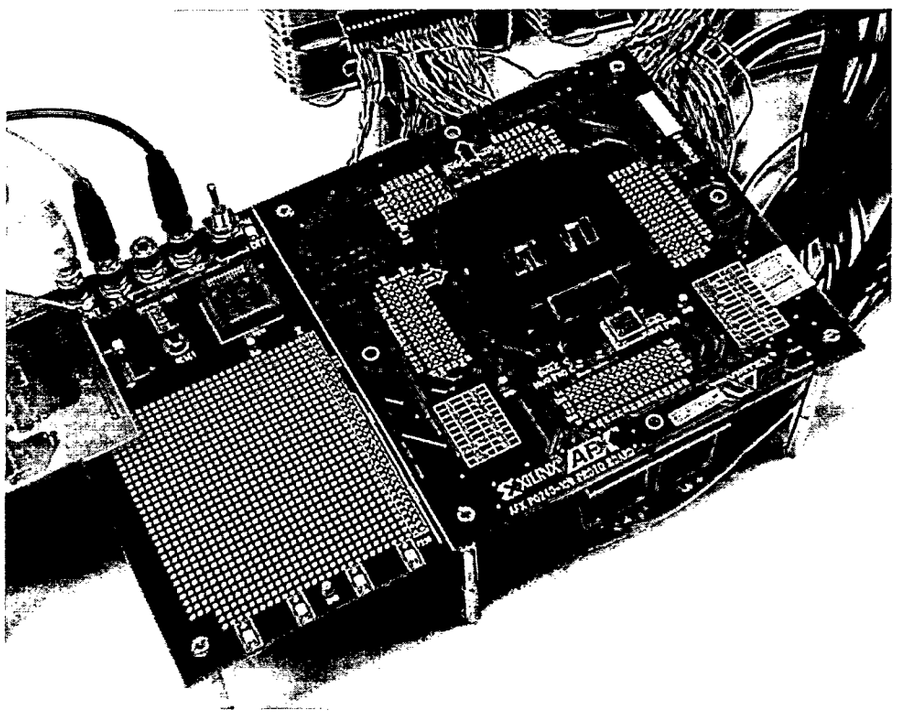



Fig. 138: Fotografía de la tarjeta AFX

Una vez decidida la tarjeta que se emplearía, había que diseñar una interfaz para poder manejarla a través de JBits. Se optó por la solución más sencilla, el puerto paralelo del PC. Otras soluciones, como diseñar una tarjeta PCI, hubieran sido mucho más complejas, y sólo habrían implicado una mejora en la velocidad de configuración, algo que no es crítico en aplicaciones térmicas, pues las constantes de tiempo son grandes. Siguiendo esta filosofía de sencillez, se optó por emplear la interfaz SelectMap para programar la FPGA. El puerto JTAG, a parte de mucho más lento, es más complicado de manejar.

La interfaz se basa en el modo bidireccional de funcionamiento del puerto paralelo, de esta manera se puede hacer readbacks de 8 bits en paralelo de una manera muy sencilla. En la Fig. 140 se puede ver el esquemático de la interfaz diseñada. Siguiendo con el prerrequisito de sencillez, se implementó en un PCB de simple cara; su aspecto final puede verse en la siguiente figura:

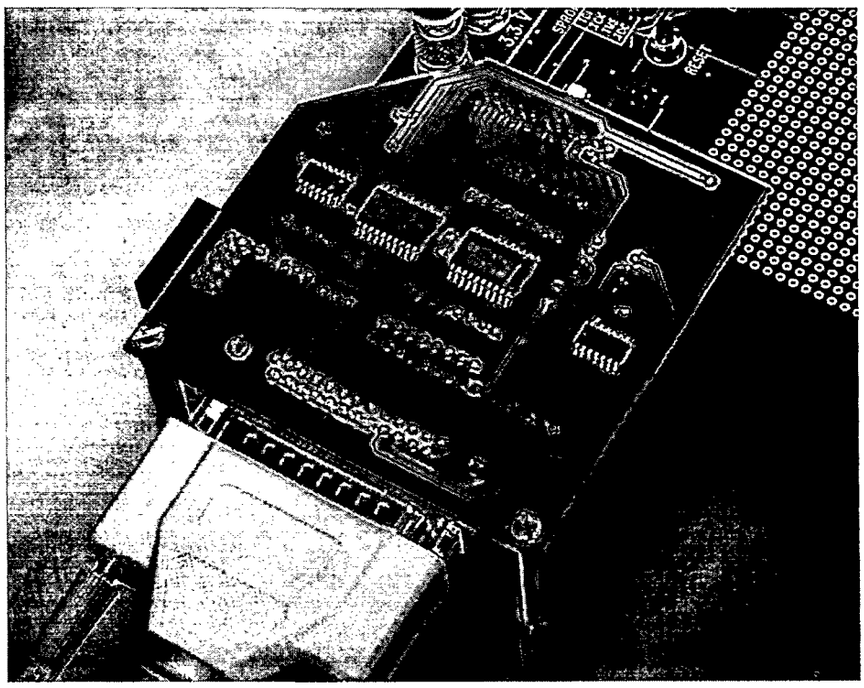


Fig. 139: Fotografía de la interfaz con la placa AFX

Básicamente, la interfaz está compuesta por un buffer bidireccional (U1) para los datos, cuya dirección está controlada por el pin de lectura/escritura del puerto SelectMap; así se elimina el riesgo de colisión. Las pines CS, CCLK, RW y PROG del SelectMap se manejan a través de las salidas de control del puerto paralelo; como esas salidas son de colector abierto, se han utilizado búferes con histéresis (U3) para eliminar problemas de ruido, pues la subida de 0 a 1 se realiza lentamente a través de un pull-up de 1K2, y si no se usaran estos búferes pueden producirse varias transiciones durante la subida, lo cual para la señal CCLK es desastroso. Este valor del pull-up, y los de las resistencias de terminación, se han obtenido del estándar IEEE1284. Las señales que se leen desde la FPGA (DONE, INIT, BUSY) pasan también a través de un búfer (U2).

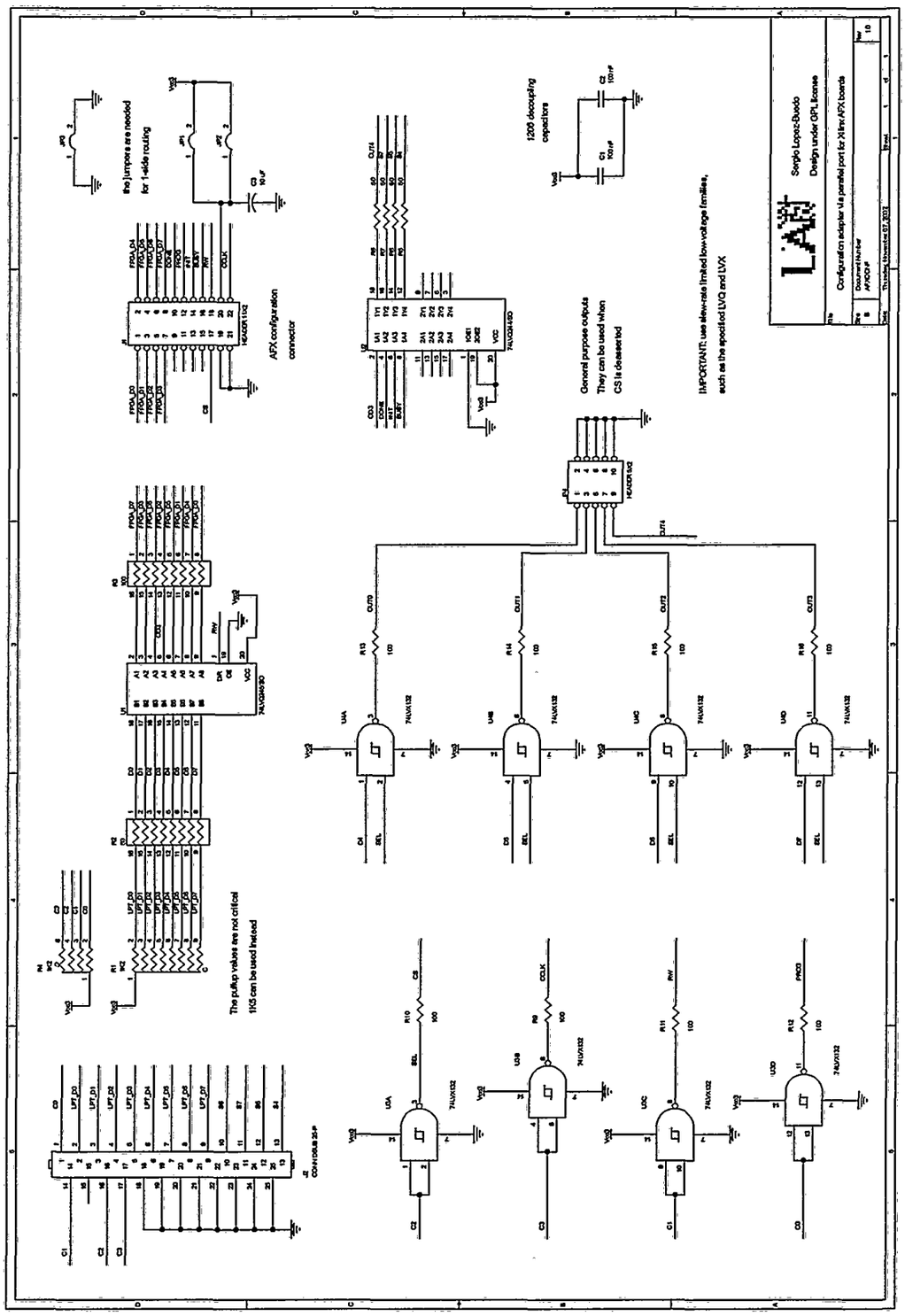



Fig. 140: Esquemático de la interfaz con la placa AFX

Adicionalmente se ha habilitado un conector con 5 salidas de propósito general. Las cuatro primeras se corresponden con D4 hasta D7, siempre que no se esté configurando la FPGA (SEL, o sea, CS a 1); y la quinta es D3 pasada por un búfer. La utilidad de estas señales podría ser generar relojes paso a paso, o cualquier otra señal para controlar el circuito que se haya implementado en la FPGA. Para generar estas señales se utiliza U4 y un búfer libre de U2. Se ha utilizado otra puerta NAND con histéresis (74LVX132) para simplificar y no emplear componentes distintos, pero en este caso la histéresis no es estrictamente necesaria.

Por último, hay que mencionar que se han utilizado familias de bajo slew-rate para evitar problemas de ruido (LVX y LVQ), más teniendo en cuenta que se ha utilizado un PCB de simple cara.

# 3. Código JBits del sensor

El sensor está codificado como un RTPCore, y por tanto implementa de los métodos necesarios en cualquier core:

- Los constructores de la clase, que en este caso hay dos. El primero es el normal, sólo con una entrada: el reloj del sistema. El segundo permite tener acceso a la señal de salida del oscilador en anillo, a través de un puerto adicional de salida. Esta segunda alternativa será sólo útil para depuración, porque sacar esta señal fuera incrementará el autocalentamiento.
- Unos sencillos métodos que se emplean para calcular el tamaño y la granularidad del core; esta es una práctica común en todos los RTPCores.
- El método `implement`, que será el que generará en sí el circuito.

La generación del circuito se ha encapsulado en dos métodos privados. El primero, `MakeConnections`, se encarga sólo de crear y rutar los nodos: define la fuente y los destinos de las señales, y luego llama a `Bitstream.connect` para crear la ruta con la JRoute, la herramienta de rutado automático de JBits. Por otro lado, el método `ConfigureCLBs` encapsula toda la creación de la lógica, consistiendo en llamadas a `Bitstream.set` para ir configurando cada uno de los recursos programables de los CLBs.

```
/*****************************************************************************************************/
/*                                                                                                   */
/* Temperature Sensor RTP Core                                                                       */
/*                                                                                                   */
/* Sergio Lopez-Buedo, Microelectronics Lab @ Computer Eng. School, Universidad Autonoma de Madrid   */
/*                                                                                                   */
/* v0.1 (initial), August 2001                                                                       */
/* v0.2, April 2003, minor changes because it did not work on the space left by an anticore          */
/*                                                                                                   */
/*****************************************************************************************************/

import com.xilinx.JBits.CoreTemplate.*;
import com.xilinx.JRoute2.Virtex.ResourceDB.CenterWires;
import com.xilinx.JBits.Virtex.Bits.*;
import com.xilinx.JBits.Virtex.Util;
import com.xilinx.JBits.Virtex.Expr;
import com.xilinx.JBits.Virtex.JBits;

public class TemperatureSensor extends RTPCore
{
    /*************************************************************************/
    /*                                                                       */
    /* private data                                                          */
    /*                                                                       */
    /*************************************************************************/

    /* input port for the clock */
    private Port clkPort;
    /* output port for monitoring the ring oscillator frequency */
    /* only for debugging. not recommended because of its self-heating problems */
    private Port outPort;

    /*************************************************************************/
    /*                                                                       */
    /* constructors, normal and ring-oscillator debugging                    */
    /*                                                                       */
    /*************************************************************************/

    /* normal constructor, with just one port: the clock input */
    public TemperatureSensor(String name, Net _clk) throws CoreException
    {
        super(name);

        /* create the clock port */
        clkPort = newInputPort("CLK", _clk);

        /* specify that the output port will not be used */
        outPort = null;

        /* Set the Height and Width of the Core */
        setHeight(calcHeight());
        setWidth(calcWidth());

        /* Set the Height and Width Gran of the Core */
        setHeightGran(calcHeightGran());
        setWidthGran(calcWidthGran());

    } /* normal constructor */

    /* debugging constructor: includes an output for the ring-oscillator */
    public TemperatureSensor(String name, Net _clk, Net _out) throws CoreException
    {
        super(name);

        /* create the clock port */
        clkPort = newInputPort("CLK", _clk);

        /* create the output port for monitoring ring-oscillator frequency */
        outPort = newOutputPort("OUT", _out);

        /* Set the Height and Width of the Core */
        setHeight(calcHeight());
        setWidth(calcWidth());

        /* Set the Height and Width Gran of the Core */
        setHeightGran(calcHeightGran());
        setWidthGran(calcWidthGran());

    } /* ring-oscillator debugging constructor */
```

```
/*****************************************************************************/
/*                                                                           */
/* convention methods                                                        */
/*                                                                           */
/*****************************************************************************/

public static int calcHeight()
{
    return 8;
}

public static int calcWidth()
{
    return 1;
}

public static int calcHeightGran()
{
    return Gran.CLB;
}

public static int calcWidthGran()
{
    return Gran.CLB;
}

public final void implement(int[] TimeEnable, int[] RingEnable, int[] CountEnable)
                throws CoreException
{
    int row;
    int col;
    Offset  offset;

    /* calculate the core's offset */
    offset = calcAbsoluteOffset();
    col = offset.getHorOffset(Gran.CLB);
    row = offset.getVerOffset(Gran.CLB);

    MakeConnections(row,col);
    ConfigureCLBs(row,col,TimeEnable,RingEnable,CountEnable);
}


/*****************************************************************************/
/*                                                                           */
/* private methods                                                           */
/*                                                                           */
/*****************************************************************************/

private void MakeConnections(int row, int col) throws CoreException
{
    /* auxiliary index variable for loops */
    int i;

    /* pins of the timing counter */
    Pin[]   TimeCnt_CK_Pins = new Pin[8];
    Pin[]   TimeCnt_XQ_Pins = new Pin[7];
    Pin[]   TimeCnt_F1_Pins = new Pin[7];
    Pin[]   TimeCnt_YQ_Pins = new Pin[7];
    Pin[]   TimeCnt_G1_Pins = new Pin[7];
    Pin[]   TimeCnt_SR_Pins = new Pin[7];
    Pin[]   TimeCnt_CE_Pins = new Pin[7];

    /* pins of the ring counter */
    Pin[]   RingCnt_CK_Pins = new Pin[8];
    Pin[]   RingCnt_XQ_Pins = new Pin[7];
    Pin[]   RingCnt_F1_Pins = new Pin[7];
    Pin[]   RingCnt_YQ_Pins = new Pin[7];
    Pin[]   RingCnt_G1_Pins = new Pin[7];
    Pin[]   RingCnt_SR_Pins = new Pin[7];
    Pin[]   RingCnt_CE_Pins = new Pin[7];

    /* pins for the logic that generates the clock enable of the timing counter */
    Pin TimeCnt_CE_Sink1;
    Pin TimeCnt_CE_Sink2;
    Pin TimeCnt_CE_Sink3;
    Pin TimeCnt_CE_Sink4;
    Pin TimeCnt_CE_Source;
```

```
/* pins for the logic that generates the clock enable of the ring counter */
Pin RingCnt_CE_Sink1;
Pin RingCnt_CE_Sink2;
Pin RingCnt_CE_Sink3;
Pin RingCnt_CE_Sink4;
Pin RingCnt_CE_Source;

/* pins for the logic that generates the enable of the ring-oscillator */
Pin RingEnable_Sink1;
Pin RingEnable_Sink2;
Pin RingEnable_Sink3;
Pin RingEnable_Sink4;
Pin RingEnable_Source;

/* source of the signal that resets both counters */
Pin Global_SR_Source;

/* pins of the ring-oscillator */
Pin Ring1_Source;
Pin Ring1_Sink;
Pin Ring2_Source;
Pin Ring2_Sink;
Pin Ring3_Source;
Pin Ring3_Sink;
Pin Ring4_Source;
Pin Ring4_Sink;
Pin Ring5_Source;
Pin Ring5_Sink;
Pin Ring6_Source;
Pin Ring6_Sink;
Pin Ring7_Source;
Pin Ring7_Sink1;
Pin Ring7_Sink2;

/* enable input of the ring-oscillator */
Pin RingEnable_Sink;

/* output of the ring-oscillator, is the clock for the ring counter */
Pin RingCnt_CK_Source;

/*****************************************************************************/
/*                                                                           */
/* control signals: clock enables and reset signal for both counters         */
/*                                                                           */
/*****************************************************************************/

/* clock enable for the timing counter */
TimeCnt_CE_Source = new Pin(Pin.CLB, row+7, col, CenterWires.Slice_XQ[0]);
TimeCnt_CE_Sink1 = new Pin(Pin.CLB, row+7, col, CenterWires.SliceF1[0]);
TimeCnt_CE_Sink2 = new Pin(Pin.CLB, row+7, col, CenterWires.SliceF2[0]);
TimeCnt_CE_Sink3 = new Pin(Pin.CLB, row+7, col, CenterWires.SliceF3[0]);
TimeCnt_CE_Sink4 = new Pin(Pin.CLB, row+7, col, CenterWires.SliceF4[0]);
TimeCnt_CK_Pins[7] = new Pin(Pin.CLB, row+7, col, CenterWires.SliceClk[0]);

/* clock enable for the ring counter */
RingCnt_CE_Source = new Pin(Pin.CLB, row+7, col+1, CenterWires.Slice_XQ[0]);
RingCnt_CE_Sink1 = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceF1[0]);
RingCnt_CE_Sink2 = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceF2[0]);
RingCnt_CE_Sink3 = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceF3[0]);
RingCnt_CE_Sink4 = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceF4[0]);
RingCnt_CK_Pins[7] = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceClk[0]);

/* ring enable */
RingEnable_Source = new Pin(Pin.CLB, row+7, col+1, CenterWires.Slice_Y[0]);
RingEnable_Sink1 = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceG1[0]);
RingEnable_Sink2 = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceG2[0]);
RingEnable_Sink3 = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceG3[0]);
RingEnable_Sink4 = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceG4[0]);

/* global (both counters) reset signal */
Global_SR_Source = new Pin(Pin.CLB, row+7, col, CenterWires.Slice_YQ[0]);
```

```
/*****************************************************************************/
/*                                                                           */
/* create the pins for the timing counter and the ring counter               */
/*                                                                           */
/*****************************************************************************/

for(i=0; i<7; i++)
{
    /* pins for the timing counter */
    TimeCnt_XQ_Pins[i] = new Pin(Pin.CLB, row+i, col, CenterWires.Slice_XQ[0]);
    TimeCnt_F1_Pins[i] = new Pin(Pin.CLB, row+i, col, CenterWires.SliceF1[0]);
    TimeCnt_YQ_Pins[i] = new Pin(Pin.CLB, row+i, col, CenterWires.Slice_YQ[0]);
    TimeCnt_G1_Pins[i] = new Pin(Pin.CLB, row+i, col, CenterWires.SliceG1[0]);
    TimeCnt_SR_Pins[i] = new Pin(Pin.CLB, row+i, col, CenterWires.SliceSR[0]);
    TimeCnt_CE_Pins[i] = new Pin(Pin.CLB, row+i, col, CenterWires.SliceCE[0]);
    TimeCnt_CK_Pins[i] = new Pin(Pin.CLB, row+i, col, CenterWires.SliceClk[0]);

    /* pins for the ring counter */
    RingCnt_XQ_Pins[i] = new Pin(Pin.CLB, row+i, col+1, CenterWires.Slice_XQ[0]);
    RingCnt_F1_Pins[i] = new Pin(Pin.CLB, row+i, col+1, CenterWires.SliceF1[0]);
    RingCnt_YQ_Pins[i] = new Pin(Pin.CLB, row+i, col+1, CenterWires.Slice_YQ[0]);
    RingCnt_G1_Pins[i] = new Pin(Pin.CLB, row+i, col+1, CenterWires.SliceG1[0]);
    RingCnt_SR_Pins[i] = new Pin(Pin.CLB, row+i, col+1, CenterWires.SliceSR[0]);
    RingCnt_CE_Pins[i] = new Pin(Pin.CLB, row+i, col+1, CenterWires.SliceCE[0]);
    RingCnt_CK_Pins[i] = new Pin(Pin.CLB, row+i, col+1, CenterWires.SliceClk[0]);
}

/* make the feedback connections, so that the counter can work */
for(i=0; i<7; i++)
{
    /* timing counter */
    Bitstream.connect(TimeCnt_XQ_Pins[i], TimeCnt_F1_Pins[i]);
    Bitstream.connect(TimeCnt_YQ_Pins[i], TimeCnt_G1_Pins[i]);

    /* ring counter */
    Bitstream.connect(RingCnt_XQ_Pins[i], RingCnt_F1_Pins[i]);
    Bitstream.connect(RingCnt_YQ_Pins[i], RingCnt_G1_Pins[i]);
}

/* connect the CE and SR signals for the timing counter */
Bitstream.connect(TimeCnt_CE_Source, TimeCnt_CE_Pins);
Bitstream.connect(Global_SR_Source, TimeCnt_SR_Pins);

/* connect the CE and SR signals for the ring counter */
Bitstream.connect(RingCnt_CE_Source, RingCnt_CE_Pins);
Bitstream.connect(Global_SR_Source, RingCnt_SR_Pins);

/* connect the signals that generate the clock enable for the timing counter */
Bitstream.connect(TimeCnt_XQ_Pins[5], TimeCnt_CE_Sink1);
Bitstream.connect(TimeCnt_YQ_Pins[5], TimeCnt_CE_Sink2);
Bitstream.connect(TimeCnt_XQ_Pins[6], TimeCnt_CE_Sink3);
Bitstream.connect(TimeCnt_YQ_Pins[6], TimeCnt_CE_Sink4);

/* connect the signals that generate the clock enable for the ring counter */
Bitstream.connect(TimeCnt_XQ_Pins[5], RingCnt_CE_Sink1);
Bitstream.connect(TimeCnt_YQ_Pins[5], RingCnt_CE_Sink2);
Bitstream.connect(TimeCnt_XQ_Pins[6], RingCnt_CE_Sink3);
Bitstream.connect(TimeCnt_YQ_Pins[6], RingCnt_CE_Sink4);

/* connect the signals that generate the clock enable for the ring counter */
Bitstream.connect(TimeCnt_XQ_Pins[5], RingEnable_Sink1);
Bitstream.connect(TimeCnt_YQ_Pins[5], RingEnable_Sink2);
Bitstream.connect(TimeCnt_XQ_Pins[6], RingEnable_Sink3);
Bitstream.connect(TimeCnt_YQ_Pins[6], RingEnable_Sink4);

/* trick to have the first carry input always at one*/
Bitstream.connect(Global_SR_Source, new Pin(Pin.CLB, row, col, CenterWires.SliceBX[0]));
Bitstream.connect(Global_SR_Source, new Pin(Pin.CLB, row, col+1, CenterWires.SliceBX[0]));


/*****************************************************************************/
/*                                                                           */
/* ring-oscillator related signals                                           */
/*                                                                           */
/*****************************************************************************/

/* now, create the pins for the nets that make up the ring-oscillator */
Ring1_Source = new Pin(Pin.CLB, row, col+1, CenterWires.Slice_X[1]);
Ring1_Sink = new Pin(Pin.CLB, row+2, col+1, CenterWires.SliceF1[1]);
```

```
        Ring2_Source = new Pin(Pin.CLB, row+2, col+1, CenterWires.Slice_X[1]);
        Ring2_Sink = new Pin(Pin.CLB, row+5, col+1, CenterWires.SliceF1[1]);
        Ring3_Source = new Pin(Pin.CLB, row+5, col+1, CenterWires.Slice_X[1]);
        Ring3_Sink = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceF1[1]);
        Ring4_Source = new Pin(Pin.CLB, row+7, col+1, CenterWires.Slice_X[1]);
        Ring4_Sink = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceG1[1]);
        Ring5_Source = new Pin(Pin.CLB, row+7, col+1, CenterWires.Slice_Y[1]);
        Ring5_Sink = new Pin(Pin.CLB, row+5, col+1, CenterWires.SliceG1[1]);
        Ring6_Source = new Pin(Pin.CLB, row+5, col+1, CenterWires.Slice_Y[1]);
        Ring6_Sink = new Pin(Pin.CLB, row+2, col+1, CenterWires.SliceG1[1]);
        Ring7_Source = new Pin(Pin.CLB, row+2, col+1, CenterWires.Slice_Y[1]);
        Ring7_Sink1 = new Pin(Pin.CLB, row, col+1, CenterWires.SliceF1[1]);
        Ring7_Sink2 = new Pin(Pin.CLB, row, col+1, CenterWires.SliceG1[1]);

        /* this pin is the enable input of the ring-oscillator */
        RingEnable_Sink = new Pin(Pin.CLB, row+7, col+1, CenterWires.SliceG2[1]);

        /* this pin is the source for the clock of the ring counter */
        /* it is just Ring7 buffered */
        RingCnt_CK_Source = new Pin(Pin.CLB, row, col+1, CenterWires.Slice_Y[1]);

        /* connect the nets that make up the ring-oscillator */
        Bitstream.connect(Ring1_Source, Ring1_Sink);
        Bitstream.connect(Ring2_Source, Ring2_Sink);
        Bitstream.connect(Ring3_Source, Ring3_Sink);
        Bitstream.connect(Ring4_Source, Ring4_Sink);
        Bitstream.connect(Ring5_Source, Ring5_Sink);
        Bitstream.connect(Ring6_Source, Ring6_Sink);
        Bitstream.connect(Ring7_Source, Ring7_Sink1);
        Bitstream.connect(Ring7_Source, Ring7_Sink2);

        /* the ring-oscillator is enabled by the same signal that enables the timing counter */
        Bitstream.connect(RingEnable_Source, RingEnable_Sink);

        /* route the clock for the ring counter */
        Bitstream.connect(RingCnt_CK_Source, RingCnt_CK_Pins);

        /* route the clock for the timing counter */
        clkPort.setPin(TimeCnt_CK_Pins);

        /* if the debugging output port is enabled, route it */
        if (outPort != null) outPort.setPin(Ring4_Source);

    } /* MakeConnections */


    private void ConfigureCLBs(int row, int col, int[] TimeEnable, int[] RingEnable, int[] CountEnable)
                throws CoreException
    {
        /* auxiliary index variable for loops */

        int i;


        /*****************************************************************************/
        /*                                                                           */
        /* CLB configuration for both counters (they are identical)                  */
        /*                                                                           */
        /*****************************************************************************/

        for(i=0; i<7; i++)
        {
            /***********************/
            /* timing counter, LE0 */
            /***********************/

            /* set the F LUT Contents */
            Bitstream.set(row+i, col, LUT.SLICE0_F, Util.InvertIntArray(Expr.F_LUT("F1")));

            /* CarrySelect mux for the carry propagation */
            Bitstream.set(row+i, col, S0Control.XCarrySelect.XCarrySelect,
                          S0Control.XCarrySelect.LUT_CONTROL);

            /* FOUT_XOR_CARRY to XDin input of the X Flip Flop */
            Bitstream.set(row+i, col, S0Control.X.X, S0Control.X.FOUT_XOR_CARRY);
            Bitstream.set(row+i, col, S0Control.XDin.XDin, S0Control.XDin.X);

            /* enable reset signal */
            Bitstream.set(row+i, col, S0Control.XffSetResetSelect, S0Control.OFF);
```

```
/**********************/
/* timing counter, LE1 */
/**********************/

/* set the G LUT Contents */
Bitstream.set(row+i, col, LUT.SLICE0_G, Util.InvertIntArray(Expr.G_LUT("G1")));

/* CarrySelect mux for the carry propagation */
Bitstream.set(row+i, col, S0Control.YCarrySelect.YCarrySelect,
              S0Control.XCarrySelect.LUT_CONTROL);

/* GOUT_XOR_CARRY to YDin input of the Y Flip Flop */
Bitstream.set(row+i, col, S0Control.Y.Y, S0Control.Y.GOUT_XOR_CARRY);
Bitstream.set(row+i, col, S0Control.YDin.YDin, S0Control.YDin.Y);

/* enable reset signal */
Bitstream.set(row+i, col, S0Control.YffSetResetSelect, S0Control.OFF);

/*********************************************/
/* timing counter, settings common to both LEs */
/*********************************************/

/* Cin mux for the carry input, but in the first stage we use BX */
if (i!=0) Bitstream.set(row+i, col, S0Control.Cin.Cin, S0Control.Cin.CIN);
else Bitstream.set(row+i, col, S0Control.Cin.Cin, S0Control.Cin.BX);

/* AndMux for the carry propagation */
Bitstream.set(row+i, col, S0Control.AndMux.AndMux, S0Control.AndMux.ZERO);

/* latch mode OFF */
Bitstream.set(row+i, col, S0Control.LatchMode, S0Control.OFF);

/* async reset */
Bitstream.set(row+i, col, S0Control.Sync, S0Control.OFF);

/* this two lines are also necessary just in case the bitstream is not clean */
/* that is, a previous configuration already exists on the Slice */
/* new in v0.2, because of anticore problems */
Bitstream.set(row+i, col, S0Control.SrWeNotInvert, S0Control.OFF);
Bitstream.set(row+i, col, S0Control.CeInvert, S0Control.OFF);


/*********************/
/* ring counter, LE0 */
/*********************/

/* set the F LUT Contents */
Bitstream.set(row+i, col+1, LUT.SLICE0_F, Util.InvertIntArray(Expr.F_LUT("F1")));

/* CarrySelect mux for the carry propagation */
Bitstream.set(row+i, col+1, S0Control.XCarrySelect.XCarrySelect,
              S0Control.XCarrySelect.LUT_CONTROL);

/* GOUT_XOR_CARRY to XDin input of the X Flip Flop */
Bitstream.set(row+i, col+1, S0Control.X.X, S0Control.X.FOUT_XOR_CARRY);
Bitstream.set(row+i, col+1, S0Control.XDin.XDin, S0Control.XDin.X);

/* enable reset signal */
Bitstream.set(row+i, col+1, S0Control.XffSetResetSelect, S0Control.OFF);

/*********************/
/* ring counter, LE1 */
/*********************/

    /* set the G LUT Contents */
Bitstream.set(row+i, col+1, LUT.SLICE0_G, Util.InvertIntArray(Expr.G_LUT("G1")));

/* CarrySelect mux for the carry propagation */
Bitstream.set(row+i, col+1, S0Control.YCarrySelect.YCarrySelect,
              S0Control.XCarrySelect.LUT_CONTROL);

/* GOUT_XOR_CARRY to YDin input of the Y Flip Flop */
Bitstream.set(row+i, col+1, S0Control.Y.Y, S0Control.Y.GOUT_XOR_CARRY);
Bitstream.set(row+i, col+1, S0Control.YDin.YDin, S0Control.YDin.Y);

/* enable reset signal */
Bitstream.set(row+i, col+1, S0Control.YffSetResetSelect, S0Control.OFF);
```

```
    /*******************************************/
    /* ring counter, settings common to both LEs */
    /*******************************************/

    /* Cin mux for the carry input, but in the first stage we use BX */
    if (i!=0) Bitstream.set(row+i, col+1, S0Control.Cin.Cin, S0Control.Cin.CIN);
    else Bitstream.set(row+i, col+1, S0Control.Cin.Cin, S0Control.Cin.BX);

    /* AndMux for the carry propagation */
    Bitstream.set(row+i, col+1, S0Control.AndMux.AndMux, S0Control.AndMux.ZERO);

    /* latch mode OFF */
    Bitstream.set(row+i, col+1, S0Control.LatchMode, S0Control.OFF);

    /* async reset */
    Bitstream.set(row+i, col+1, S0Control.Sync, S0Control.OFF);

    /* this two lines are also necessary just in case the bitstream is not clean */
    /* that is, a previous configuration already exists on the Slice */
    /* new in v0.2, because of anticore problems */
    Bitstream.set(row+i, col+1, S0Control.SrWeNotInvert, S0Control.OFF);
    Bitstream.set(row+i, col+1, S0Control.CeInvert, S0Control.OFF);

} /* for */


/*****************************************************************************/
/*                                                                           */
/* CLB configuration for the control signals (clock enables and resets)      */
/*                                                                           */
/*****************************************************************************/

/*****************************/
/* CE for the timing counter */
/*****************************/

/* set the F LUT contents */
Bitstream.set(row+7, col, LUT.SLICE0_F, Util.InvertIntArray(TimeEnable));

/* FOUT to XDin input of the X Flip Flop */
Bitstream.set(row+7, col, S0Control.X.X, S0Control.X.FOUT);
Bitstream.set(row+7, col, S0Control.XDin.XDin, S0Control.XDin.X);

/* disable the CE signal */
Bitstream.set(row+7, col, S0CE.S0CE, S0CE.OFF);

/* disable the SR signal */
Bitstream.set(row+7, col, S0SR.S0SR, S0SR.OFF);

/* this line is also necessary just in case the bitstream is not clean */
/* that is, a previous configuration already exists on the Slice */
/* new in v0.2, because of anticore problems */
Bitstream.set(row+7, col, S0Control.SrWeNotInvert, S0Control.OFF);

/* latch mode OFF */
Bitstream.set(row+7, col, S0Control.LatchMode, S0Control.OFF);

/***********************/
/* global reset signal */
/***********************/

/* set the G LUT contents */
Bitstream.set(row+7, col, LUT.SLICE0_G, Util.InvertIntArray(Expr.G_LUT("0")));

/* GOUT to the Y output of the slice*/
Bitstream.set(row+7, col, S0Control.Y.Y, S0Control.Y.GOUT);
Bitstream.set(row+7, col, S0Control.YDin.YDin, S0Control.YDin.Y);

/***************************/
/* CE for the ring counter */
/***************************/

/* set the F LUT contents */
Bitstream.set(row+7, col+1, LUT.SLICE0_F, Util.InvertIntArray(CountEnable));

/* FOUT to XDin input of the X Flip Flop */
Bitstream.set(row+7, col+1, S0Control.X.X, S0Control.X.FOUT);
Bitstream.set(row+7, col+1, S0Control.XDin.XDin, S0Control.XDin.X);
```

```
        /* disable the CE signal */
        Bitstream.set(row+7, col+1, S0CE.S0CE, S0CE.OFF);

        /* disable the SR signal */
        Bitstream.set(row+7, col+1, S0SR.S0SR, S0SR.OFF);

        /* this line is also necessary just in case the bitstream is not clean */
        /* that is, a previous configuration already exists on the Slice */
        /* new in v0.2, because of anticore problems */
        Bitstream.set(row+7, col+1, S0Control.SrWeNotInvert, S0Control.OFF);

        /* latch mode OFF */
        Bitstream.set(row+7, col+1, S0Control.LatchMode, S0Control.OFF);

        /***************************/
        /* ring-oscillator enable  */
        /***************************/

        /* set the F LUT contents */
        Bitstream.set(row+7, col+1, LUT.SLICE0_G, Util.InvertIntArray(RingEnable));

        /* FOUT to XDin input of the X Flip Flop */
        Bitstream.set(row+7, col+1, S0Control.Y.Y, S0Control.Y.GOUT);


        /*****************************************************************************/
        /*                                                                           */
        /* CLB configuration for the ring oscillator                                 */
        /*                                                                           */
        /*****************************************************************************/

        /* Ring1 */
        Bitstream.set(row, col+1, LUT.SLICE1_F, Util.InvertIntArray(Expr.F_LUT("~F1")));
        Bitstream.set(row, col+1, S1Control.X.X, S1Control.X.FOUT);

        /* Ring2 */
        Bitstream.set(row+2, col+1, LUT.SLICE1_F, Util.InvertIntArray(Expr.F_LUT("~F1")));
        Bitstream.set(row+2, col+1, S1Control.X.X, S1Control.X.FOUT);

        /* Ring3 */
        Bitstream.set(row+5, col+1, LUT.SLICE1_F, Util.InvertIntArray(Expr.F_LUT("~F1")));
        Bitstream.set(row+5, col+1, S1Control.X.X, S1Control.X.FOUT);

        /* Ring4 */
        Bitstream.set(row+7, col+1, LUT.SLICE1_F, Util.InvertIntArray(Expr.F_LUT("~F1")));
        Bitstream.set(row+7, col+1, S1Control.X.X, S1Control.X.FOUT);

        /* Ring5, this one has the ring enable input */
        Bitstream.set(row+7, col+1, LUT.SLICE1_G, Util.InvertIntArray(Expr.G_LUT("~G1&G2")));
        Bitstream.set(row+7, col+1, S1Control.Y.Y, S1Control.Y.GOUT);

        /* Ring6 */
        Bitstream.set(row+5, col+1, LUT.SLICE1_G, Util.InvertIntArray(Expr.G_LUT("~G1")));
        Bitstream.set(row+5, col+1, S1Control.Y.Y, S1Control.Y.GOUT);

        /* Ring7 */
        Bitstream.set(row+2, col+1, LUT.SLICE1_G, Util.InvertIntArray(Expr.G_LUT("~G1")));
        Bitstream.set(row+2, col+1, S1Control.Y.Y, S1Control.Y.GOUT);

        /* RingCnt_CK_Source */
        Bitstream.set(row, col+1, LUT.SLICE1_G, Util.InvertIntArray(Expr.G_LUT("~G1")));
        Bitstream.set(row, col+1, S1Control.Y.Y, S1Control.Y.GOUT);

    } /* ConfigCLBs */

}
```

# Barbarismos

En esta sección se ha tratado de hacer un glosario de los términos que, pese a no estar reconocidos por la R.A.E., están plenamente integrados en el lenguaje de los ingenieros, al menos en el entorno del autor. Todos ellos provienen del idioma inglés.

**autocalentamiento** m. Traducción del término *self-heating*. En esta tesis se refiere al calentamiento que sufre un sensor de temperatura por la potencia que disipa durante su operación.

**búfer** m. Castellanización del término *buffer*. No existe ninguna palabra en español que reúna la variedad de significados de este término: amplificador y restaurador de niveles (electrónica digital), memoria intermedia (programación), etc...

**bus** m. Castellanización del mismo término inglés. Se incluye porque su significado según el diccionario no tiene nada que ver con el que se le da en electrónica: colección de señales. En cualquier caso, el origen de estas palabra y de otras muchas en varios idiomas viene del término latino *omnibus*, "para todos".

**core** m. Castellanización del mismo término inglés. La traducción más aproximada, núcleo, no se corresponde con la acepción más común en el mundo de las FPGAs: circuito prediseñado.

**glitch** m. Castellanización del mismo término inglés. Muy empleado al no existir un término español tan compacto; lo más aproximado sería transición espuria, que resulta ligeramente cacofónico.

**fanout** m. Castellanización del mismo término inglés. No existe una palabra española para expresar este concepto: conexiones o carga de una señal.

**flip-flop** m. Castellanización del mismo término inglés. Aunque existe una palabra castellana con similar significado, biestable, en el entorno del autor no es usada. Por esta razón es discutible su uso, pero se incluye en esta lista porque ha terminado desplazando al término español.

**grapinar** v. t. Castellanización y conversión en verbo del término *wrapping*. Utilizado en el contexto de *wire-wrapping*, esto es, crear prototipos de circuitos con conexiones de hilo arrollado.

**hardware** m. Castellanización del mismo término inglés. Palabra muy empleada en el lenguaje habitual, no sólo en la jerga informática, pues no existe un término español similar.

**instancia** m. Castellanización del término *instance*. Aunque este término existe en el diccionario, ninguna de sus acepciones se corresponde con el significado que tiene tanto en diseño electrónico como en programación orientada a objetos: elemento correspondiente a una determinado tipo o clase.

**instanciar** v. t. Castellanización del verbo *to instantiate*. Su significado sería crear un nuevo elemento de un determinado tipo o clase; no existe ningún verbo español que tenga todos estos matices.

**layout** m. Castellanización del mismo término inglés. No existe ninguna palabra española con el mismo significado en el área de microelectrónica: disposición de los elementos que componen un circuito.

**macro** f. Aunque este término existe en el diccionario, en él aparece como prefijo que significa "grande". Sin embargo, en el contexto de esta tesis tiene el significado de circuito prediseñado.

**mapear** v. t. Castellanización del verbo *to map*. En el contexto de esta tesis se podría traducir como asociar recursos a elementos lógicos; no sería correcto emplear sólo el verbo asociar, porque no tiene todos los matices de *to map*.

**metaestabilidad** m. Adaptación del término *metaestability*. Por su raíz completamente latina, esta plenamente justificado su uso.

**pad** m. Castellanización del mismo término inglés. No existe una palabra en español que tenga exactamente los mismos matices, pues no es la pata de un circuito integrado, sino el área donde se conecta

**pin** m. Castellanización del mismo término inglés. En español se utiliza el término pata de un circuito integrado, pero es cierto que esta palabra es también muy usada.

**reconfigurar** v. t. Adaptación del término *reconfigure*. Proviene de una raíz latina, por lo que no debería considerarse muy pernicioso su uso.

**reconfigurable** adj. Adaptación del término *reconfigurable*. Al igual que el término anterior, no debería considerarse muy pernicioso su uso por provenir de una raíz latina.

**reset** m. Castellanización del término *reset*. La palabra equivalente en español, reiniciación, es mucho menos compacta, por eso ha sido desplazada por este barbarismo.

**resetear** v. t. Castellanización del verbo *to reset*. Su uso está causado por lo común del término reset antes mencionado, pero es discutible por la existencia de un verbo español muy compacto: reiniciar.

**rutar** v. t. Castellanización del verbo *to route*. Las acepciones que aparecen en el diccionario son completamente distintas al uso que se le da en esta tesis, crear un camino para establecer las conexiones de una señal.

**temporización** m. Castellanización del término *temporization*. En español existen muchas construcciones similares; la única salvedad es que el significado que da en el diccionario a temporizar no es corresponde con el concepto para el que se emplea en esta tesis: fijar la evolución en el tiempo de algo.

**software** m. Castellanización del mismo término inglés. Palabra muy empleada en el lenguaje habitual, no sólo en la jerga informática, pues no existe un término español similar.

**triestado** adj. Castellanización del término *triestate*. Proviene de una raíz latina, por lo que no debería considerarse muy pernicioso su uso.

# Bibliografía


**[Aav02]** Aavid Thermalloy LLC., *"New Cooling Solutions for Ball Grid Arrays: Product Selection Guide"*, abril 2002.

**[Act03]** Actel Inc., *"Package Characteristics and Mechanical Drawings"*, febrero 2003.

**[Alf97]** P. Alfke, *"Contribución al grupo de noticias comp.arch.fpga"*, abril 1997.

**[Alf98]** P. Alfke, *"Comunicación personal con E. Boemo"*, 8 de junio 1998.

**[Alt98a]** P. Alfke, *"Comunicación personal con E. Boemo"*, 10 de junio 1988.

**[Alt99]** P. Alfke, *"Comunicación personal con E. Boemo"*, 12 de marzo de1999.

**[Alt99a]** P. Alfke, *"Comunicación personal con E. Boemo"*, 16 de marzo de 1999.

**[Alt01]** J. Altet, A. Rubio, E. Schaub, S. Dilhaire y W. Claeys, "Thermal Coupling in Integrated Circuits: Application to Thermal Testing", *IEEE Journal of Solid-State Circuits*, Vol. 36, No. 1, enero 2001.

**[Alt02]** J. Altet, S. Dilhaire, S. Volz, J.-M. Rampnoux, A. Rubio, S. Grauby, L. D. Patino, W. Claeys, J.-B. Saulnier, "Four different approaches for the measurement of IC surface temperature: application to thermal testing", *Microelectronics Journal*, No. 33, pp. 689-696, 2002.

**[Alt02b]** Altera Inc., *"Reliability Report No. 37"*, segundo cuatrimestre de 2002.

**[Alt03]** Altera Inc., *"Altera Device Package Information"*, febrero 2003.

**[AMD03]** Advanced Micro Devices, Inc. *"AMD AthlonXP Processor Model 10 Datasheet"*, febrero 2003.

**[Asu02]** Asustek Computer Inc., *"V9280 Graphics Card User's Manual"*, octubre 2002.

**[Atm99]** Atmel Corp., *"AT6000 Series Datasheet"*, octubre 1999.

**[Atm03]** Atmel Corp., *"AT40K05/10/20/40AL Complete Datasheet"*, febrero 2003

**[Boe97]** E. Boemo y S. Lopez-Buedo, "Themal Monitoring on FPGAs using Ring-Oscillators", *Lecture Notes in Computer Science*, No. 1304, pp.69-78, Berlin: Springer-Verlag, 1997.

**[Dag98]** J. M. Daga, E. Ottaviano y D. Auvergne, "Temperature Effect on Delay for Low Voltage Applications", *Proc. DATE'98 (Design, Automation and Test in Europe)*, pp. 680-685, París, febrero 1998.

**[Del02]** Delphi Technologies Inc., "*PST3-02 Thermal Test Die*", Julio 2002.

**[Ell95]** Gordon Ellison, "Fan cooled enclosure analysis using a first order method", *Electronics Cooling*, octubre 1995.

**[Fau97]** Julio Faura, Juan Manuel Moreno, Miguel Angel Aguirre, Phuoc van Duong y Josep Maria Insenser, "Multicontext Dynamic Reconfiguration and Real-Time Probing on a Novel Mixed Signal Programmable Device with On-Chip Microprocessor", *Lecture Notes in Computer Science*, No. 1304, pp. 1-10, Berlin: Springer-Verlag, 1997.

**[Gig97]** Giga Operations Corp., *"XMOD Features"*, 1997.

**[HMP99]** Hybrid Memory Products Ltd., *"SRAM Module Mean Time Between Failure Analysis"*, febrero 1999.

**[Int98]** Intel Corp., *"Introduction to Thermal Solutions"*, diciembre 1998.

**[Int01]** Intel Corp., *"Intel ® Pentium ® 4 Processor in the 423-pin Package at 1.30, 1.40, 1.50, 1.60, 1.70, 1.80, 1.90 and 2 GHz Datasheet"*, agosto 2001.

**[Jac97]** M. Jackson, P. Lall y D. Das, "Thermal Deration - A Factor of Safety or Ingnorance", *IEEE Trans. on Components, Packaging and Manufacturing Technology,* Part A, Vol.20, No.1, pp.83-85, marzo 1997.

**[JED95]** Electronic Industries Association, *"Integrated Circuits Thermal Test Method Environmental Conditions - Natural Convection (Still Air)"*, EIA/JEDEC Standard JESD51-2, diciembre 1995.

**[JED96]** Electronic Industries Association, *"Low Effective Thermal Conductivity Test Board For Leaded Surface Mount Packages"*, EIA/JEDEC Standard JESD51-3, agosto 1996.

**[JED97]** Electronic Industries Association, *"Thermal Test Chip Guideline (Wire Bond Type Chip)"*, EIA/JEDEC Standard JESD51-4, febrero 1997.

**[JED99]** Electronic Industries Association, *"High Effective Thermal Conductivity Test Board For Leaded Surface Mount Packages"*, EIA/JEDEC Standard JESD51-7, febrero 1999.

**[JED99b]** Electronic Industries Association, *"Integrated Circuit Thermal Test Method Environmental Conditions - Forced Convection (Moving Air)"*, EIA/JEDEC Standard JESD51-6, marzo 1999.

**[JED01]** JEDEC Solid State Technology Association, *"Failure Mechanisms and Models for Semiconductor Devices"*, JEDEC Publication JEP122-A, diciembre 2001.

**[Lal96]** P. Lall, "Tutorial: Temperature as an Input to Microelectronics-Reliability Models", *IEEE Trans. on Reliability*, Vol.45, No.1, pp.3-9, March 1996.

**[Las97]** C. Lasance, "Thermal Resistance: An Oxymoron?", *Electronics Cooling*, mayo 1997.

**[Lop97]** S. Lopez-Buedo and E. Boemo, "A Method for Temperature Measurement on Reconfigurable Systems", *Proc. XII DCIS Conference (Design of Circuit and Integrated Systems)*, pp.727-730, Universidad de Sevilla: noviembre 1997.

**[Lop98]** S. Lopez-Buedo, J. Garrido y E. Boemo, "Thermal Testing on Programmable Logic Devices", *1998 IEEE ISCAS (Int. Symp. on Circuits and Systems)*, Vol.II, pp.240-243, Monterey, June 1998.

**[Lop00]** S. Lopez-Buedo, J. Garrido y E. Boemo, "Thermal Testing on Reconfigurable Computers", *IEEE Design & Test of Computers,* pp.84-90, enero-marzo 2000.

**[Mac98]** E. Macii, M. Pedram y F. Somenzi, "High-Level Power Modeling, Estimation, and Optimization", *Proc. Computer-Aided Design of Integrated Circuits and Systems*, 1998.

**[Max99]** Maxim Integrated Products, *"MAX1619 Remote/Local Temperature Sensor with Dual-Alarm Outputs and SMBus Serial Interface"*, abril 1999.

**[Mic01]** MicReD Ltd., *"T3Ster, The Thermal Transient Tester. Main System Unit and Add-on Options Overview"*, 2001

**[MIL96]** Departamento de Defensa de los E.E.U.U, *"Thermal characteristics of Integrated Circuits"*, U.S. Military Standard MIL-STD 883E, diciembre 1996.

**[Naj94]** F.N.Najm, "A Survey of Power Estimation Techniques in VLSI Circuits", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 2, pp. 446-455, diciembre 1994.

**[NIS02]** NIST y SEMATECH, *"E-Handbook of Statistical Methods"*, capítulo 8, disponible en *http://www.itl.nist.gov/div898/handbook/*, 2002.

**[Pab01]** Thomas Pabst y Frank Völkel, *"How Modern Processors Cope With Heat Emergencies"*, Tom's Hardware Guide, septiembre 2001.

**[Pec97]** Michael G. Pecht, "Open Forum Editorial", *IEEE Trans. on Components, Packaging and Manufacturing Technology,* Part A, Vol.20, No.1, pp.83, marzo 1997.

**[Ped97]** M. Pedram, "Design Technologies for Low Power VLSI", *Encyclopedia of Computer Science and Technology*, Vol. 36, pp. 73-96, Marcel Dekker, Inc., 1997.

**[Poo02]** Kara K.W. Poon, Andy Yan y Steven J.E. Wilton, "A Flexible Power Model for FPGAs", *Lecture Notes in Computer Science*, No. 2438, pp. 312-321, Berlín: Springer-Verlag, 2002.

**[Que91]** G. Quenot, N. Paris y B. Zavidovique, "A temperature and voltage measurement cell for VLSI circuits", *Proc. 1991 EURO ASIC Conf.*, pp-334-338, IEEE Press, 1991.

**[Rob00]** David Robinson y Patrick Lysaght, "Verification of Dynamically Reconfigurable Logic", *Lectures Notes in Computer Science*, No. 1896, Berlin: Springer-Verlag 2000.

**[Rho01]** Steve Rhoads, *"The Plasma CPU Core"*, disponible en *opencores.org*, septiembre 2001.

**[SIA00]** Semiconductor Industry Association, *"International Technology Roadmap for Semiconductors, 2000 Update"*, 2000.

**[Sip99]** M. Sipper, D. Mange y E. Sanchez, "Quo Vadis Evolvable Hardware?", *Communications of the ACM*, Vol. 42, No. 4, pp. 50-56, April 1999.

**[SEM87]** Semiconductor Equipment and Materials International, "*Test Method for Junction-to-Case Thermal Resistance Measurements of Molded Plastic Packages*", SEMI Standard G43-87, 1987.

**[SEM88]** Semiconductor Equipment and Materials International, *"Test Method for Junction-to-Case Thermal Resistance Measurements of Ceramic Packages"*, SEMI Standard G30-88, 1988.

**[Sun03]** Prasanna Sundararajan, *"Contribución al grupo de discusión sobre JBits"*, disponible en el foro JBits de *groups.yahoo.com*, 17 de abril de 2003.

**[Sut89]** I. Sutherland, "Micropipelines", *Communications of the ACM*, Vol. 22, No. 6, pp.720-734. junio 1989.

**[Sut02]** Gustavo Sutter, *"Comunicación personal con S. López"*, diciembre 2002.

**[Sya01]** A. Syal, V. Lee, A. Ivanov, J. Altet, "CMOS Differential and Absolute Thermal Sensors", *Proceedings 7th On-Line Testing Workshop*, pp. 127-132, 2001.

**[Sze95]** V. Székely, Cs. Márta, M. Rencz, y B. Courtois, "A New Approach: Design for Thermal Testability (DfTT) of MCMs", *Proc. MCM Test Workshop*, pp.1-3, Napa Valley, 1995.

**[Sze97]** V. Székely, Cs. Márta, Zs. Kohári, M. Rencz, "CMOS Sensors for On-Line Thermal Monitoring of VLSI Circuits", *IEEE Transactions on VLSI Systems*, Vol. 5, No. 3, pp. 270-276, septiembre 1997.

**[The99]** Thermacore Inc., *"Graphic Processor Thermal Control Design Guide"*, agosto 1999.

**[TI99]** Texas Instruments Inc., *"Package Thermal Characterization Methodologies (Application Report)"*, marzo 1999.

**[Tiw98]** Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel, Franklin Baez, "Reducing Power in High-performance Microprocessors", *Proc. 35th Design Automation Conference*, San Francisco, California, 1998.

**[Tod02]** E. Todorovich, M. Gilabert, G. Sutter, S. López-Buedo y E. Boemo, "A Tool for Activity Estimation in FPGAs", *Lecture Notes in Computer Science*, No. 2438, pp. 340-349, Berlín: Springer-Verlag, 2002.

**[Tra02]** Transmeta Corp., *"Crusoe TM5500/TM5800 Thermal Design Guide"*, julio 2002.

**[Tur96]** Mike Turner, "All you need to know about fans", *Electronics Cooling*, Vol.2, No. 2, mayo 1996.

**[Vcc99]** Virtual Computer Corp., *"VW-300. The Virtual Workbench"*, disponible en http://www.vcc.com.

**[Xes01]** Xess Corp., *"XSV Board V1.0 Manual"*, mayo 2001.

**[Xil95]** Xilinx Inc., *"Programmable Logic Breakthrough '95. Technical Conference and Seminar Series"*, pp. 6-11, 1995.

**[Xil96]** Xilinx Inc., *"Trading Off Among the Three Ps: Power, Package and Performance"*, XCELL No. 22, 1996.

**[Xil97]** Xilinx Inc., *"Xilinx Series 6000 User Guide"*, 1997.

**[Xil99]** Xilinx Inc., *"Virtex Analog to Digital Converter"*, nota de aplicación 155, septiembre 1999.

**[Xil99b]** Xilinx Inc., *"Libraries Guide. Chapter 8"*, incluida en la documentación *on-line* de la herramienta DOCSAN, 1999.

**[Xil99c]** Xilinx Inc., *"Status and Control Semaphore Registers Using Partial Reconfiguration"*, nota de aplicación 153, junio 1999.

**[Xil99d]** Xilinx Inc., *"Xilinx Prototype Platforms User Guide for Virtex and Virtex-E Series FPGAs"*, diciembre 1999.

**[Xil02]** Xilinx Inc., *"Packaging Thermal Management"*, nota de aplicación 415, julio 2002.

**[Xil02b]** Xilinx Inc., *"The Reliability Data Program, Expanded Version"*, octubre 2002.

**[Xil02c]** Xilinx Inc., *"What is the tolerance of the internal clock of XC4000/XC5200 devices?"*, Answer Record No. 499, Answers Database, mayo 2002.

**[Xil02d]** Xilinx Inc., *"Virtex FPGA Series Configuration and Readback"*, nota de aplicación 138, julio 2002.

**[Xil02e]** Xilinx Inc., *"Configuration and Readback of Virtex FPGAs Using JTAG Boundary Scan"*, nota de aplicación 139, abril 2002.

**[Xil02f]** Xilinx Inc., *"Two Flows for Partial Reconfiguration:Module Based or Small Bit Manipulations"*, nota de aplicación 290, mayo 2002.

**[Xil02g]** Xilinx Inc., *"Development System Reference Guide – ISE 5"*, 2002.

**[Xil03]** Xilinx Inc., *"Virtex Series Configuration Architecture User Guide"*, nota de aplicación 151, marzo 2003.

Reunido el tribunal que suscribe en el día
de la fecha, acordó calificar la presente Tesis
doctoral con SOBRESALIENTE CUM LAUDEM
Madrid, 27 de Junio de 2008

Jesus José Serrano Martín

Eduardo Sánchez

Josep Rius Vázquez

Javier Valls Coquillat

Antonio García Ríos